The Automatic Assessment of Multiple Artefacts: An Investigation into Design Diagrams and Their Implementations

Thesis

The Automatic Assessment of Multiple Artefacts: An Investigation into Design Diagrams and Their Implementations

Alan Michael Hayes B.A. (hons), M.Sc.

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science

Department of Computing

Faculty of Mathematics, Computing and Technology

The Open University

December 2013

Abstract

As the Higher Education sector has moved towards student-centred learning so too has the growth in electronic support for learning. E-assessment has been a part of this growth as increasingly assessment and its feedback is seen as an integral part of the students' learning process. Mature e-assessment systems exist, particularly where answers to questions are restricted to a prescribed list of alternatives. However, for free response artefacts, where there is a limited restriction placed on answers to questions, automated assessment systems are embryonic.

This dissertation presents an investigation into the automated assessment of free response artefacts. Design diagrams and their accompanying source code implementations are examples of free response artefacts. A case study is developed that investigates how to automatically generate formative feedback for a design diagram by utilizing its accompanying implementation. The dissertation presents a two-staged solution, initially analysing the design diagram in isolation before comparing it with the implementation. A framework for this approach has been developed and tested using a tool applied to coursework submitted by undergraduate computer science students.

The tool was evaluated by comparing the formative feedback comments generated by the tool with those produced by a team of computer science educators. Evaluation was undertaken via two Likert questionnaires, one completed by students and one completed by a team of computer scientists. The results presented are favourable, with the majority of comments produced by the tool being seen to be as least as good as those generated by the computer science educators.

Author's Declaration

All of the work presented in this dissertation describes original contributions of the author (exceptions noted on the Acknowledgements page). Some of the material in this dissertation has been published previously in the following papers:

Hayes A., Thomas P., Smith N., Waugh K. 2007 **A Framework for the Automated Assessment of Consistency Between Code and Design.** In Proceedings of Informatics Education Conference 11, Thessaloniki, Greece November 2007.

Hayes A. 2007 **The Development of An Automated Assessment Framework [online].** In Assessment in Wales: Practice which Works: a snapshot of assessment practice in Wales. Available from **http://www.heacademy.ac.uk/resources/detail/resource_database/casestudies/welsh_case_studies_index**

Hayes A., Thomas P., Smith N., Waugh K., 2007, **An Investigation into the Automated Assessment of the Design-Code Interface.** Proceedings of the 12th Annual Conference on Innovation and Technology in Computer Science Education. Poster Presentation ITiCSE2007, University of Dundee, July 2007.

Hayes, A., Thomas, P., Smith, N., Waugh, K., 2007. **A Developmental Framework for Computer-based Automated Assessment.** Technical Report, The Open University, Ref 2006/07, ISSN 1744-1986, April 2007.

Acknowledgements

I would like to thank the following:

My family: Jayne, Anne-Marie and Jacob for their support and understanding.

My supervisors: Pete Thomas, Neil Smith and Kevin Waugh, for their patience, support, feedback, advice and encouragement. They were invaluable in helping me to keep going.

The Open University for the flexibility in enabling me to juggle the demands of part-time study with the commitments of a family and a full-time job.

The academic colleagues from the Computer Science Education Community who engaged with the evaluation phase of this research. These were:

Allan Blair - London South Bank University,

Derek Paddon – University of Bath,

Chris Sturley – Plymouth University,

Vicki Bush – University of Gloucestershire,

Claire Willis – University of Bath,

Wendy Ivins – Cardiff University,

Ann Latham – Wolverhampton University,

Christopher Tubb – University of Wales, Newport,

Torbjorn Dahl – University of Wales Newport,

Martin Stanton – Manchester Metropolitan University,

Eric Llewellyn – University of Wales, Newport,

Marianna Lilley – University of Hertfordshire.

Contents

Table of Figures

Table of Figures continued

Table of Figures continued

Tables

Chapter 1. Introduction

This dissertation addresses the problem of how to automatically generate high quality formative feedback for freeform design diagrams. It defines what constitutes good quality feedback and presents a novel and robust method for its evaluation. It documents a framework for the computer assisted assessment (CAA) of design diagrams. It identifies those core concepts and components that such a framework needs to encompass. It describes how a design diagram's source code implementation can be used to aid its assessment. It presents a method for generating formative feedback that utilises both the implementation and typical errors contained in diagrams produced by novice designers. An assessment tool is presented which implements the principles enshrined within the framework and applies them to work submitted by undergraduate students studying computer science and computing-related programmes of study. It highlights how the design/implementation context is one example of the generic case where feedback is generated for two related artefacts. The dissertation generalises throughout by presenting and defining terms for multiple artefacts and their assessment. It demonstrates the automated generation of formative feedback based upon an analysis of the two artefacts.

1.1 Motivation and Educational Context

The Higher Education (HE) Landscape in the UK is changing. Students are growing in number. From academic session 2002/03 to 2011/12 the total number of HE students (part-time and full-time) in UK Higher Education Institutions (HEIs) grew from 2,131,110 to 2,496,645. For the sciences, the rise for the same period was from 868,700 to 1,485,770 with 76,590 students studying undergraduate

computer science programmes in 2011/12. (Higher Education Statistics Agency, http://www.hesa.ac.uk).

Students pay for their tuition, with fees first being introduced in 1998/99 (initially £1000 with subsequent inflationary rises to £1,250). Variable fees of £3000 were introduced in 2006/07 (England and Northern Ireland) and 2007/08 (Wales) and a cap of £3250 was introduced in 2009/10. For HEIs in England, this cap was increased to £9,000 for new 2012/13 entrants (Bolton, 2012). Beer (2011) predicts that the market culture introduced by fees will lead to a marked increase in the level of student demand and expectation.

Students are entering University with a much more informed voice, primarily due to the advent of the National Student Survey (NSS). The survey asks final year undergraduate students to evaluate their learning experience (National Student Survey, 2012). It has been undertaken in UK Higher Education Institutions annually since its launch in 2005. The results of the survey are made public via the UNISTATS website (http://unistats.direct.gov.uk). The results have highlighted that students are less positive about assessment and feedback on their assignments than other aspects of their learning experience (Wiliams *et al.* 2008; Boud and Molloy 2012). Consequently, Higher Education Institutions are being criticised more for inadequacies in their feedback than for any other aspect of their provision (Boud and Molloy 2012). In the 2005 survey, for example, the lowest scoring items within the 'assessment and feedback' section were statement 7 'Feedback on my work has been prompt' and statement 9 'Feedback on my work has helped me clarify things I did not understand' (Williams *et al.* 2008 citing Surridge 2006). In exploring assessment and feedback issues raised in the NSS, Williams *et al.*

(2008) identify pockets of good practice across the sector. These include the provision of feedback that is prompt, timely and occurs in a range of formats.

The demographic profile of students is also changing. This is contributed to by HEIs being required to produce access agreements approved by the Office For Fair Access (OFFA). These agreements are mandatory for HEIs wishing to charge full tuition fees and are designed to ensure that everyone with the potential to benefit from higher education has an equal opportunity to do so, regardless of background, age, ethnicity, disability or gender (OFFA, http://www.offa.org.uk). The resultant diverse cohort of students poses challenges for university teachers (Laurillard, 2012). Furthermore, recent generations of students have been referred to as 'Digital Natives' or the 'Net Generation' meaning that they have grown up with computers, social networking and the internet and have a natural aptitude and skillset with information technology (Jones *et. al.* 2010). Prensky (2001) made the distinction between students who had grown up with technology (digital natives) and older educators who had not (digital immigrants), postulating that today's students are no longer the people our education system was designed to teach. Jones *et. al.* (2010) reported that the vast majority of students in their study (first year undergraduate students at 5 Universities in England) made extensive use of mobile technologies and computing facilities for access to course materials and resources.

This growth in the number of diverse, digitally-literate, survey-informed, fee-paying students has been responded to in several ways by the HE sector. One example is the adoption of technology to support both educators in their teaching and students in their learning. Technological support occurs in many ways ranging from the provision of access to on-line learning material, the submission of

coursework assignments, plagiarism detection and assessment. Technology is enabling both students and educators to do more. However, it needs to deliver systems that are scalable (to meet volume) and flexible (to meet diverse needs and expectations).

Educators need to master new technologies as they are shaping what is learnt by changing how it is learnt (Laurillard, 2012). The Open University's UK-based platform for Massive Open On-Line Courses (MOOCS) is an example of how the sector is using technology to adapt to the changing student needs and expectation. The Open University consider their platform to be "… the next chapter in the story of British Higher Education" (Parr, 2012).

One area of recent technological growth to support student learning is that of e-assessment (Joy *et al.* 2002, Terzis and Economides 2011). E-assessment offers the potential to enable educators to manage the growing number of students whilst meeting the needs of remote and mobile learners and addressing the area of assessment and feedback highlighted as a student concern by the NSS returns. However, technology to enhance assessment is embryonic (Whitelock and Watt, 2007) and ideas about its pedagogic impact are still in their infancy (Conole and Warburton 2005). Practical-based disciplines, such as computer science, require subject-specific learning support tools (Lass *et al.* 2003*)* particularly in the provision of multiple types of feedback (Iahad and Dafoulas 2004a).

Good quality formative feedback needs to be consistent, accurate, useful and timely. Feedback should positively reinforce good practice in addition to identifying where further learning is required. Feedback that emphasises mistakes and inadequacies has a negative effect upon student retention and engagement (Baker and Zuvela 2012). However, human educators can't agree on precisely

what constitutes good quality formative feedback comments (Yorke 2003). This poses a challenge for its evaluation.

Automated assessment systems can accept either free-form or fixed responses (Culwin 1998). Fixed response systems prescribe a limited range of responses available to the user, for example a multiple choice test. Free response systems allow the user much more latitude in what they submit. Examples of free form items include essays, source code and design diagrams.

Undergraduate computer science students studying software engineering explore a wide-range of techniques, tools, methodologies, design diagrams and implementation languages (Sommerville 2007). They will frequently be asked to produce these artefacts as a part of their assessment. Free form diagrams are particularly challenging for automated assessment systems. They may contain errors or extraneous data (Smith *et al.* 2004, Thomas *et al.* 2005) and their semantics are often semi-formally prescribed. Diagrams do not restrict students to a limited range of fixed responses that, for example, multiple-choice systems do i.e. their content is free-form as the student has not been curtailed to producing a diagram from a prescribed list of pre-determined solutions. Consequently, there can be many different but correct diagrams for the same assignment specification.

The research question, therefore, for this dissertation is that given the changing nature of Higher Education, how can we automatically generate high quality feedback for student design task submissions?

In addressing this question, this research makes the following contributions:

- It defines criteria for categorising automated assessment tools.
- It presents a method for automating the assessment of design diagrams by utilising both their implementations and established work that has identified known errors made by novice designers.
- It provides a definition for what constitutes high quality formative feedback and presents a novel and robust method for its evaluation.
- It presents the generic case by defining terms for multiple artefacts and their assessment.
- It documents an automated assessment tool that generates quality formative feedback.

1.2 Classification of Automated Assessment Tools

This dissertation identifies some of the core characteristics of tools that automate assessment. It proposes a categorisation of such systems according to three characteristics: the type of student submission (fixed or free form), the type of feedback generated (summative mark or formative comments) and the extent of the automation (semi or fully automated).

1.3 Automated Assessment of Diagrams

This dissertation presents a novel means to automating the formative assessment of student design diagrams. A blended approach is presented that initially searches for typical errors in the student design diagram before comparing it with its implementation. Several potential methods emerge from this approach and these are investigated.

1.4 Defining and Evaluating Good Quality Feedback

This dissertation proposes three broad criteria against which formative feedback comments can be evaluated: quality, relevance and coverage. Additionally, it presents a novel and robust method for evaluating automatically generated formative feedback. The method involves a comparison with feedback generated by a team of expert markers. It addresses variations in human markers when assessing student work and evaluating formative feedback**.**

1.5 Multiple Artefacts: the Generic Case for Diagram Comparison

Comparing a design diagram with its accompanying implementation is one example of the generic case of two artefacts referring to the same referent. These artefacts represent different, but complementary, views of a solution. Other examples include a text-based requirement specification and its formal mathematical notation or an architectural design and its building specification. In the design/implementation context, the design (in diagrammatic format) is viewed as prescribing the structure and function contained within the implementation, whilst the implementation (source code) is viewed as implementing the design whilst adhering to its specified structure and function.

This research generalises throughout by presenting and defining terms for multiple artefacts and their assessment. A method for describing an artefact's constituent features and a heuristic for their comparison is presented.

1.6 The Development of an Automated Assessment Tool

The approach is validated through the development of a proof-of-concept tool that automatically generates formative feedback comments based upon a comparison of two artefacts. It has been applied to student submitted assignments collated over several years from two Higher Education institutions. The submissions consist of a design diagram and its accompanying implementation. A grammar has

been implemented that describes the artefacts and their constituent features. The tool generates feedback that positively reinforces good practice whilst identifying where further learning is needed.

1.7 Summary and Roadmap for the remainder of the Dissertation

In summary, automating feedback is challenging and is one example of how technology can be used to meet the changing profile and expectation of UK higher education students. Students are growing in number, digitally literate and are accustomed to remote and mobile access to learning. Freeform diagrams are particularly challenging for automated assessment systems as they may contain errors or extraneous data and their semantics are often semi-formally prescribed. They do not restrict students to a limited range of fixed responses. The research question addressed by this dissertation is:

Given the changing nature of Higher Education, how can we automatically generate high quality feedback for student design task submissions in the form of diagrams?

The remainder of this dissertation is structured as follows:

Chapter 2 reviews the literature from which the research question was refined and developed. It identifies that the assessment of a design diagram and its accompanying implementation has not, to my knowledge, been addressed by existing automated diagram assessment tools.

Chapter 3 recognises that a design diagram and its accompanying implementation is one instance of a generic case where two artefacts represent different means of expressing a solution to a problem. It provides a framework for how artefacts can be analysed and formative feedback generated.

Chapter 4 reports upon the development of a proof-of-concept tool that takes the multiple artefact concept discussed in chapter 3 and applies it to the example case where the artefacts are represented by a design diagram and its accompanying implementation.

Chapter 5 reports upon how the feedback generated by the developed tool was evaluated. It documents how the evaluation was undertaken by both students and computer science educators.

Chapter 6 reports upon the results of applying the tool to a corpus of student work.

Chapter 7 reports upon the conclusions of the research and the identification of future work.

Chapter 2. E-assessment and Diagrams

2.1 Introduction

This chapter focuses on E-assessment and highlights its limited application to diagrams. In particular, for diagrams produced by undergraduate computer science students, we know of no systems that automate the assessment of both a design diagram and its accompanying implementation. Design diagrams and their implementations are examples of free-form artefacts. Automating their assessment is challenging. Automatically generating formative feedback for multiple free-form artefacts is new and brings together several different fields and is the focus for this research project. This chapter positions the research within these fields.

This chapter presents the motivating educational context. It discusses the general principles of e-assessment providing definitions for fixed, free-form summative and formative assessment (section 2.2). It proposes a method for categorising automated assessment systems (section 2.3). It provides an overview of existing automated diagram assessment systems and from this identifies five core challenges that such systems address (section 2.4). It provides an overview of how existing systems have been evaluated and discusses work that has analysed and identified common errors contained in design diagrams (section 2.5). It concludes by scoping how the remainder of this research will progress and how it has been informed by the literature (section 2.6).

2.2 E-assessment a Definition, its Growth and Perceived Benefits.

This section defines what an e-assessment system is. It discusses how the electronic tools contained in such systems support both the assessment of student work and the administrative processes surrounding such assessment. It outlines

how recent growth in the adoption of e-assessment by the HE sector has stimulated the development of technological supportive tools and pedagogic models that incorporate their use.

The Joint Information Systems Committee's (JISC) report on Effective Practice with E-Assessment (2007) defines e-assessment to be

".. the end-to-end electronic assessment processes where ICT is used for the presentation of assessment activity, and the recording of responses. This includes the end-to-end assessment process from the perspective of learners, tutors, learning establishment, awarding bodies and regulators, and the general public." (Effective Practice with E-assessment, JISC 2007)

E-assessment encompasses the application of information and communications technology (ICT) to support activities undertaken to assess student-submitted work. The extent of the application has led to a plethora of terminology which surrounds the use of ICT in Higher Education (Bull and Danson, 2004). The number and types of processes that are automated can be used to differentiate between the terms Computer Assisted Learning (CAL), Computer Based Learning (CBL), Computer Assisted Assessment (CAA) and Computer Based Assessment (CBA), as illustrated in Figure 2.0 below (Higgins and Bligh, 2006).

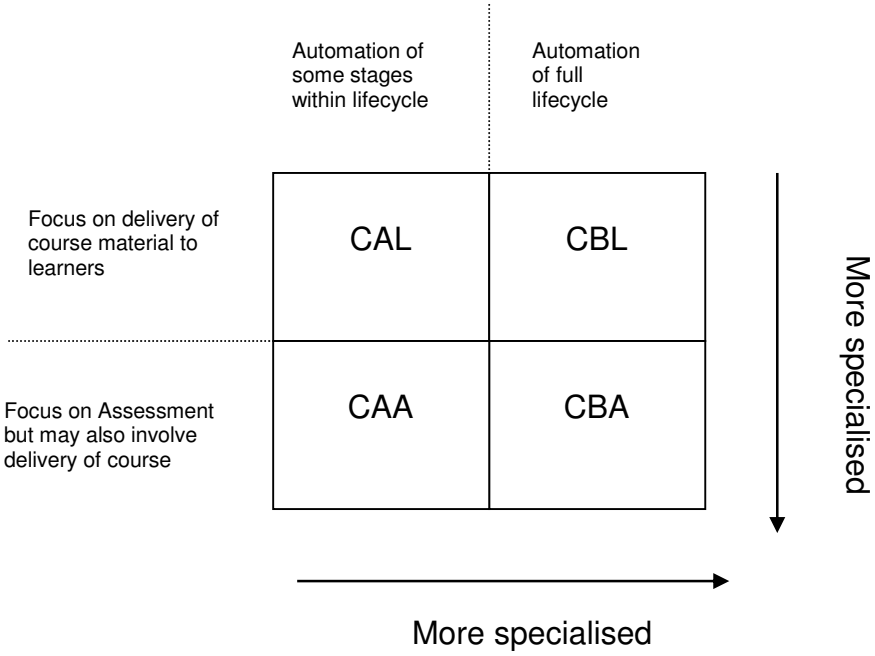


Figure 2.0: The relationship between Computer Aided Learning, Computer Based Learning, Computer Aided Assessment and Computer Based Assessment (Higgins and Bligh 2006)

The focus of both CAA and CBA includes, but is not restricted to, the application of ICT to the marking and grading of student work. The extent that electronic support extends beyond that for marking differentiates between the two. CAA is the application of computer technologies specifically to the assessment process (Bull and Danson, 2004) whilst CBA has a broader application of ICT which extends to "…delivery of materials for teaching and assessment, the input of solutions by the students, an automated assessment process and the delivery of feedback, all achieved through an integrated, coherent, online system." (Higgins and Bligh, 2006).

The use of e-assessment in the Higher Education sector has recently expanded (Joy *et al.* 2002; Terzis and Economides 2011). Its emergence has contributed to the adoption of student-centred approaches to learning and teaching with the tutor acting as a facilitator of learning (Iahad *et al.* 2004b). The benefits of e-assessment can be divided into two groups: practical and pedagogical. Practical

benefits include supporting the delivery, marking and analysis of assignments and examinations, plagiarism detection, the recording of achievement through the construction of on-line portfolios and the transfer of assessment information over distributed networks (Bull and Danson 2004, Tselonis 2008). The practical advantages of CBA over traditional forms of assessment include engendering a fairness and consistency across the marking of a cohort's submission (Tsintsifas 2002), test security, cost and time reduction, speed of results, automatic record keeping and support for distance learning (Prados *et al.* 2011).

Tsintsifas (2002) identified three important pedagogic characteristics of an automated diagram assessment system:

- Repeatable; when a student exercise is submitted to the marking system with the same inputs, it will always receive the same mark.
- Consistent; the state of the marking system is the same both before and after marking a student's exercise.
- Reliable; when the student exercise is submitted it is guaranteed that a mark will be produced for the student.

Tsintsifas (2002)

Higgins and Bligh (2006) looked at the pedagogic benefits of CBA by considering how it met Brown *et al.'s* (1996) 10 pedagogic criteria for measuring the quality of assessment (Table 2.0). They concluded that in 7 of the 10 criteria CBA is "likely to present a distinct pedagogic advantage over traditional assessment." Higgins and Bligh (2006).

Brown *et al.*'s (1996) pedagogic criteria	Higgins and Bligh's (2006) consideration for the criteria's application to CBA
Valid	Will measure specified coursework aspects assuming good initial assessment design.
Reliable	The same assessment process will run for each submission; consistency is absolute
Fair	Design-dependent: CBA has no inherent advantages
Equitable	The same assessment process will run for each submission; discrimination is non-existent
Formative	CBA provides a good opportunity to run assessment frequently throughout the learning process, and to provide multiple submissions with full feedback each time
Timely	CBA provides a good opportunity to run assessment frequently throughout the learning process
Incremental	Design-dependent: CBA has no inherent advantages
Redeemable	CBA is suited to allowing multiple submissions should the designer wish this
Demanding	Design-dependent: CBA has no inherent advantages
Efficient	Considerable time and other resource savings to be made; originally a motivator for CBA's development.

Table 2.0 Higgins and Bligh's(2006) mapping of Brown *et al.'s* (1996) pedagogic criteria to CBA

The effective development of CBA depends upon it being accepted by the students with ease of use and perceived playfulness having a direct effect upon its take-up (Terzis and Economides 2011). Additionally, the level at which the student is studying impacts on the type and nature of ICT-based assessment tools that can be adopted. Bloom's original taxonomy (1956) classifies learning into six cognitive levels (knowledge, comprehension, application, analysis, synthesis and evaluation). The taxonomy represents an increasing level of learning abstraction and difficulty ranging from memory recall (knowledge) through to making critically informed judgements (evaluation). Most web-based

CAA/CAL tools tend to focus upon the knowledge and comprehension levels within the taxonomy resulting in systems that pattern-match user input against a tutor-supplied expected solution (Joy *et al.* 2002). Lilley *et al.* (2004) noted that in the context of Computer Based Testing (CBT) it is generally the same set of preset questions that is presented to all participating students irrespective of the potentially mixed ability of the student cohort. They identified the issue of high performing students being presented with one or more questions below their level of ability and conversely low performing students being presented with questions that are above their level of ability. This resulted in high performing students quickly losing interest and an increase in guess work from low performing students.

Both Joy *et al.* (2002) and Lilley *et al.* (2004) reported the development of systems that attempt to target the level of questions being asked to the level of ability of the student. Both select questions to be asked based upon the results of the student's response to previous questions. Lilley *et al.* (2004) claim their Computer Adaptive Testing (CAT) technique is at least as useful as traditional CBT-based alternatives

To summarise, e-assessment embraces many aspects of the traditional assessment process including the development and provision of electronic tools that support assessment administration and the marking and grading of student assignments and examinations. The use of e-assessment systems has grown within the HE sector, stimulating the emergence of technological tools to support assessment processes and pedagogic models that adopt these tools. Further work is needed in both of these areas. The focus of this research is upon the development of a CAA system.

2.3 Conceptual Categories for E-assessment Systems

This section proposes a means of categorising systems that automate the marking of a student submission. Categorisation is helpful as it enables the organization and eases the communication of existing systems and their needs. Systems that automate the marking of a student submission can be categorised in several ways. This section proposes a means of categorisation in accordance with three characteristics. These are: the type of feedback they generate, the type of input data they respond to and the extent to which the assessment is automated. This section provides an overview of each of these conceptual categories and concludes by illustrating how they can be applied to a sample of existing automated assessment systems.

The Joint Information Systems Committee's report on effective Practice with E-assessment (2007) identifies three stages at which e-assessment provides learning support. The stages are diagnostic, formative and summative. Diagnostic assessment assesses the student's knowledge prior to enrolment on a programme of study. Formative assessment is defined as providing developmental feedback to a student on current understanding and skills. Summative assessment is defined as being the final assessment of a student's achievement. Dafoulas (2005) defines summative assessment as measuring what the student has learnt and formative assessment as supporting the student to learn.

Systems that automate the assessment of diagrams generate formative or summative feedback (or both). Automated assessment systems attempt to emulate feedback that is similar to that of a human marker for both the

summative and formative case. They often provide support for the input from the tutor, typically in the form of a tutor-produced model solution, against which a comparison with the student submission can be made. This research focuses upon the development and evaluation of an automated assessment system that produces formative feedback.

The second proposed conceptual category is the type of input data the system responds to, either free-form or fixed responses (Culwin 1998). Fixed response systems prescribe a limited range of responses available to the user, for example a multiple choice test. Free response systems allow the user much more latitude in what they submit. Examples of free form items include essays, source code and design diagrams.

Questions requiring free response answers are considered to require deeper cognitive processing (Jordan 2011). Prados *et al.* (2011) observe that most of the current CBA systems only assess fixed-response questions and Culwin (1998) acknowledges that the assessment of free responses is much more difficult than fixed. Jackson (2000), Daly and Waldron (1999), Joy and Luck (1998) and Joy *et al.* (2005) are examples of the automated assessment of free response source code. The Open University's OpenMark system supports the automated assessment of free text in addition to a range of fixed-form question types including multiple-choice, multiple-response, drag-and-drop and hotspot (Jordan 2011).

Diagrams can be considered to be either free form or fixed form items. Diagrams contain drawn elements, their connections, adornments and identifying labels. A significant aspect of free formness is the labels as these are

an essential part of the recognition of the drawn elements. Fixed response diagrams are created within an environment that significantly constrains the text that a label can contain or what can be drawn. A typical context would be a student selecting and dragging diagrammatic elements or labels from a prescribed list and dropping them into specific areas of a given diagram. Free form diagrams are created with little (partially free) or no (fully free) constraint on what the student can draw. An analogy would be that of requiring a student to produce text. Adding a word, taken from a given list, to a sentence is a fixed response as it provides little choice whereas asking the student to produce a sentence is a free form response (with significant scope for choice). The majority of currently available systems only provide fixed response (Thomas *et al.* 2012). Tselonis and Sargeant (2007) acknowledge that fully automating the assessment of free-form diagrams is a very difficult task.

The focus of this research project is upon free form diagrams. For a student studying the field of computer science this would typically be that of a Unified Modelling Language (UML) class diagram or an Entity Relationship Diagram (ERD) - (Jayal and Shepperd 2009; Higgins and Bligh 2006; Tselonis *et al.* 2005).

The third conceptual category is the extent to which the assessment is automated. The extent to which a system automates the process of assessing diagrams can be divided into two main categories: those that attempt to fully automate the assessment process and those that adopt a semi-automated approach. Fully automated systems take as input the student submission and produce feedback by analysing it. Whether summative or formative, the system automatically produces the feedback. Semi-automated systems also take as input a student submission but they produce guidance, as a consequence of

analysing the input, that helps the tutor decide upon what feedback should be given to the students. Tselonis *et al.* (2005) describe this semi-automated approach as being human-computer collaborative. Their approach is to provide a human-marker with information to aid the marking process. It is the human-marker, not the automated system, which determines the students' grades.

The research presented in this dissertation focuses upon fully automated diagram assessment systems.

Hence, factors to be taken into consideration when categorising the marking support of an e-assessment system include:

- Whether the system requires a free or fixed response from the student
- Whether the system generates formative or summative feedback (or both)
- Whether the system fully or semi-automates the assessment generated

These categories are illustrated in the diagram in Figure 2.1 below.

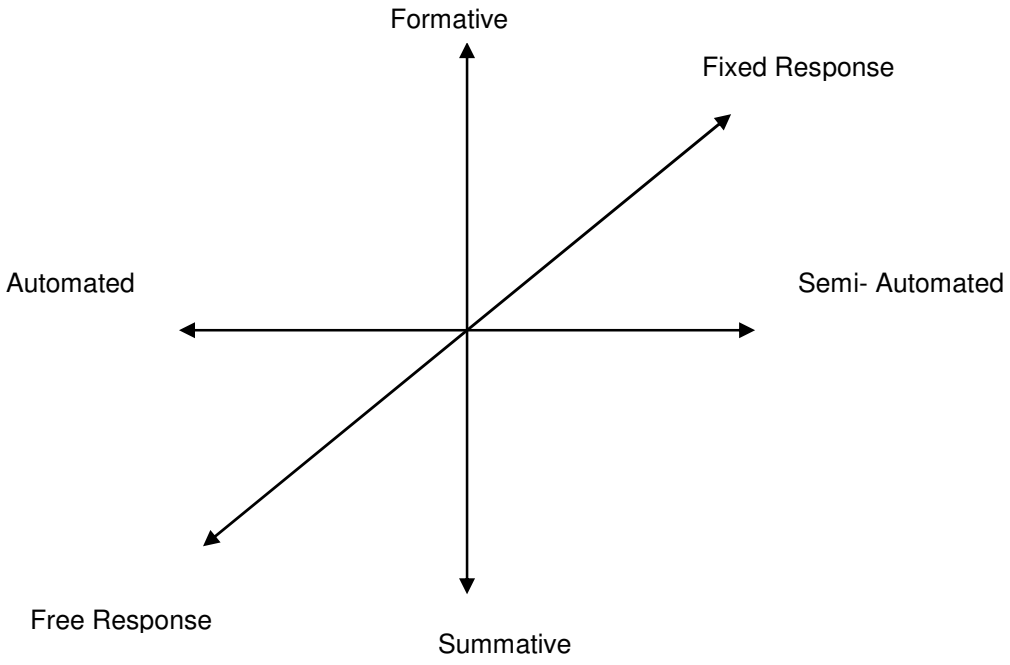


Figure 2.1: A diagram to illustrate the range of conceptual categories applicable to systems that attempt to automate the assessment of diagrams.

Five bibliographical databases were searched to identify existing diagram assessment systems. The databases, the search terms used and the number of articles returned are presented in Table 2.1. Table 2.2 maps some existing systems onto the conceptual categories identified in Figure 2.1.

	ACM Digital Library	IEEXplore	The Collection of Computer Science Bibliographies	Web of Science	Google Scholar
e-assessment of software systems for diagram-based coursework	2	96	0	0	14
"e-assessment" AND "diagrams"	9	2	7	2	4
"automated assessment" AND "diagrams"	38	5	3	4	1
"Computer Based Assessment" AND "Diagrams"	6	1	1	2	0
"Computer Aided Assessment" AND "Diagrams"	14	1	0	1	0
"Computer Based Assessment"	44	23	81	285	689
"Computer Aided Assessment"	26	24	61	110	680

Table 2.1 Bibliographical Databases searched, search terms used and number of articles returned (August 2013).

Of the systems that produce formative feedback Soler *et al.'s* (2010) feedback focussed upon the number and naming of diagram components, Ali *et al.'s* (2007a) upon the number of diagram components, Higgins and Bligh (2006) reported challenges of over-lengthy feedback comments, Higgins *et al.'s* (2009) feedback consisted of a numeric grade, Suraweera and Mitrovic (2002) offered hints on how the submission could be modified to produce the correct solution and

Hoggarth and Lockyer (1998) provided a list of deviances from a tutor-supplied model solution.

Soler *et al.* (2010) and Stone *et al.* (2009) avoid the problem of free-form labels by requiring, respectively, the student to use specific labels contained within the problem statement or importing noun-phrases from the problem statement into the diagram editor being used by the student. Consequently, both these systems have been categorised as being fixed form systems.

Most systems compare the student diagram with one or more model solutions. However, where a question allows the student a great deal of freedom it becomes difficult to enumerate all possible solutions. In an attempt to overcome this problem, Striewe and Goedicke (2011) use a set of matching rules to specify the elements that should appear in a student's diagram and those that should not. It is up to the instructor to construct the rules for each specific problem. This means that the instructor must identify the errors that are present in a given set of student diagrams prior to automatic marking. Another difficulty with this approach is that a rule which specifies that a particular label should appear must list the acceptable synonyms for that label which is not practicable with unconstrained free-text labels.

		Automated				Semi-Automated			
		Formative		Summative		Formative		Summative	
		Fixed	Free	Fixed	Free	Fixed	Free	Fixed	Free
		AFFi	AFFr	ASuFi	ASuFr	SFFi	SFFr	SSuFi	SSuFr
1	Open University DEAP (2004-2013)				■				
2	Nottingham DatSys (2002)				■				
3	Nottingham Higgins *et al.* (2009)		■						
4	Tselonis Manchester ABC								■
5	Loughborough Batmaz and Hinde (2006-2007)								■
6	Malaysia UCDA (2007)		■						
7	Spain ACME-DB (2010)	■							
8	Nottingham Higgins and Bligh (2006)		■						
9	Kermit Canterbury New Zealand (2002)		■						
10	Hogarth and Lockyear Teeside (1998)						■		
11	Loughborough Stone, Batmaz and Hinde (2009)					■			
12	Essen,Germany (2011) Striewe and Goedicke								■
13	Hayes (2013)		■						

Table 2.2 Examples of Existing Diagram Assessment Systems and their Mapping onto Three Conceptual Categories

Key

1	Thomas (2004), Thomas *et al.* (2005, 2006, 2007, 2008, 2012, 2013) Diagram Electronic Assessment Project (DEAP) Open University	2	Tsintsifas (2002) – DatSys, University of Nottingham
3	Higgins *et al.* (2009) CourseMaster, University of Nottingham	4	Tselonis *et al.* (2005) – Assess by Computer (ABC), University of Manchester
5	Batmaz and Hinde (2006; 2007) – Loughborough University	6	Ali *et al.* (2007a) – UML Class Diagram Assessor (UCDA), University Malaysia Terengganu.
7	Soler *et al.* (2010) – ACME-DB, University of Girona, Spain.	8	Higgins and Bligh (2006) CourseMarker, University of Nottingham
9	Suraweera and Mitrovic (2002) KERMIT – Canterbury (New Zealand)	10	Hoggarth and Lockyer (1998) – University of Teeside
11	Stone, Batmaz and Hinde University of Loughborough (2009)	12	Striewe and Goedicke (2011) University of Duisburg-Essen, Germany

AFFi	Automated, Formative, Fixed response
AFFr	Automated, Formative, Free response
ASuFi	Automated, Summative, Fixed response
ASuFr	Automated, Summative, Free response
SFFi	Semi-Automated, Formative, Fixed response
SFFr	Semi-Automated, Formative, Free response
SSuFi	Semi-Automated, Summative, Fixed response
SSuFr	Semi-Automated, Summative, Free response

Suraweera and Mirtovic (2004) describe a knowledge-based entity-relationship modelling tutor that uses constraint-based modelling to model domain knowledge. The domain, ERDs, is described by a set of constraints which is capable of recognising correctly formed ERDs (syntax). Each constraint specifies a fundamental property of a domain that must be satisfied by any

correct solution. The system contains a model solution for each of its problems, which is compared against the student's solution according to the system's knowledge base. The domain knowledge of KERMIT is represented as a set constraints used for testing the student's solution for syntax errors and comparing it to the ideal solution. Currently KERMIT's knowledge base consists of 92 constraints. It is well known that knowledge acquisition is a very slow, labour intensive and time consuming process. The problem of free-form labels is avoided by forcing the student to highlight the word or phrase that is modelled by each object in the ER diagram. There has to be a different set of constraints for each domain.

None of the above systems address the assessment of a design diagram and its accompanying implementation. Most generate feedback utilising input from the tutor (semi-automated) or via a comparison between the student diagram and a model solution provided by the tutor (fully automated). However, whilst using a model solution enables a summative judgement on the student diagram to be made, it does not offer formative support as the student's learning moves from high to low levels of abstraction. A system that automatically generates formative feedback based upon a comparison between the student's design and its implementation will support the student as he/she moves through the abstraction layers. The development of such a system is the focus of this research project.

To summarise, there are several approaches that can be taken to automating the assessment of student submissions. This section has identified three characteristics that can be used to classify them. These are the scope of the automation (semi or fully automated), the type of feedback generated (formative, summative or both) and the type of input that is supported (free or

fixed form). Examples of existing diagram assessment systems have been classified using these characteristics.

The research presented in this dissertation investigates the development of a framework that supports the characteristics of fully automated assessment taking as its input free form diagrams with their accompanying implementations and generating formative feedback.

2.4 Diagrams and their Assessment

Analysing student produced diagrams can be an invaluable means of assessing knowledge (Tselonis 2008). Using diagrams as a teaching tool can aid a student's learning and comprehension (Butcher and Kintsch 2004). Despite this, only a few e-assessment systems support the assessment of free form diagrams (Tselonis 2008; Prados 2011, Thomas *et al.* 2012) . This section discusses the challenges associated with automating the assessment of diagrams. The focus is on those diagrams that illustrate relationships between objects. Maps and sketches, for example, are outside the scope of this research. A review of the literature is presented and is structured around the identification of five core challenges. Section 2.4.1 defines what these challenges are and sections 2.4.2 to 2.4.6 discuss the salient issues of each challenge.

2.4.1 Challenges of an Automated Diagram Assessment System

This section discusses the five core challenges that are pertinent to automating the assessment of student submitted diagrams. The challenges are to develop appropriate mechanisms by which:

1. students can draw and submit diagrams using an electronic tool;
2. a tutor can supply a marking scheme and/or model solution(s) against which the student submission will be judged;

3. a student diagram is compared with a model solution/marking scheme.
4. issues of extraneous or erroneous data contained in the student diagram can be addressed;
5. feedback is given to a student based upon the diagram submitted.

Sections 2.4.2 to 2.4.6 below elaborate upon each of these challenges.

2.4.2 Challenge1: Student Production and Submission of Diagrams

This section distinguishes between the electronic support provided for the student to draw diagrams and the format used to represent them. Drawing support provided by existing diagram assessment systems is discussed as is the extent to which the resultant diagrams can be considered to be free form or fixed form. The importance of providing support for drawing both the individual components of a diagram and the links between each component is highlighted. The requirement of an automated assessment system to be able to identify both the components and their respective links is discussed.

When considering electronic support for assessing the student diagram there are two significant perspectives to consider. The first is that of the student and the second is that of the automated assessment system. The student needs support to be able to draw a diagram that uses domain-specific symbols and semantics. The automated assessment system requires the diagram to be in a form and format that facilitates an automatic analysis and assessment. Thomas (2004) recognised the distinction between the need to consider how a diagram should be represented for grading purposes and the mechanism for how a diagram should be graded.

It is not necessarily the case that the electronic format of the diagram produced by the student lends itself easily to being processed by the automated assessment system. Ali *et al.* (2007b) illustrate the issues involved in extracting

appropriate information from Rationale Rose files prior to automatic assessment. The work of Fan and Tanimoto (2007) is notable in that it has a minimal set of drawing primitives in an attempt to make both the students' and instructors' tasks easier. However, in doing so they have limited the expressive power of their system and cannot (yet) support a wide range of diagram types. Consequently, the development of an automated assessment system needs to balance the diagrammatic production requirements of the student with the analytical and processing requirements of the automated assessment system.

Typically, automated assessment systems require diagrams to have been produced from a finite set of graphical symbols. Such symbols have a semantic meaning associated with them that can be defined by the pedagogic context under which they are being produced. Moreover, the diagrams being drawn are underpinned by a methodology which typically contains rules for how these components can be linked together. The links themselves are also graphical symbols contained in the diagram. Such diagrams are common place in computer science and the automation of their assessment can be challenging (Jayal and Shepperd 2009). The symbols typically come from a specific development paradigm such as UML class diagrams or ERD diagrams. Diagrams are drawn using a tool that is either an integral component of the assessment system or the system specifies the type and range of drawing tool output that it supports. The extent to which a drawing tool enforces the rules of the underpinning methodology can be used to determine the degree to which an automated marking system can be considered to be either fixed or free form. For example, students producing diagrams using a CASE tool will normally have been forced to comply with rules contained within the underlying methodology. In this context, whilst the student is free to identify and draw

individual components (free form) the CASE tool adopted will curtail the extent to which the student can connect components through enforcing the rules of the underpinning methodology (fixed form). Tools that do not enforce syntactic correctness in their diagrams allow for freedom of expression, and many errors, and provide the opportunity to give feedback on a wider range of misunderstandings (Smith *et al.* 2013).

An example of a system that offers support for diagram production that goes beyond the methodological support delivered by a typical CASE tool is the Datsys system developed by Tsintsifas (2002). This provides a tool, referred to as Diadolos, which allows the assessor to define the fundamental graphical components to be used in the assessment. It also allows the specification of constraints that determine how such components can be connected. The students' diagrams are restricted to using these components under the specified constraints. The disadvantage of this approach is the tutor's initial time and investment in setting up the symbols and their connection constraints. The advantage is that it offers the flexibility for the tutor to vary the symbols and methodologies (for example across cohorts and years) without the need to acquire and configure a completely new tool/CASE environment. It also enables the tutor to restrict what can be drawn to those parts of the underpinning methodology currently being taught or to the particular learning outcomes that the development of the diagram is intended to assess.

Balancing between fixed and free response systems can also be found in Thomas (2004). In this study students produced a diagram when undertaking an on-line examination. Students were asked to submit diagrams which had been produced using an electronic tool. Submission of the students' work

involved sending the electronic diagram over the internet to an automated marking tool which graded the students' work. The drawing tool used was developed in-house and supported elementary components of boxes and links. Whilst the students were able to combine boxes and links in a free-form manner they were restricted to using only those graphical components supplied and supported by the tool. The analysis within the marking tool was restricted to consider only those graphical components. This offers similar pedagogic benefits as those outlined by Tsintsifas (2002). Both these systems, in placing a limited restriction upon how these components could be linked together, lend themselves significantly more towards the free form than fixed form category.

An approach to taking true free-form diagrams from the student can be found in Lank *et al.* (2000). They report upon a technique for recognising UML components from hand-drawn diagrams. Their system requires the diagrams to be produced on-line and in real-time using either a single or networked suite of smartboards. It works by building up a temporal-picture associated with the production of the diagram. The system tests for intersecting lines and the order in which they were produced to build up a picture of the individual diagrammatic components. Each component identified is then sent to a UML-specific recogniser where the individual diagrammatic components are recognised and classified. The disadvantage to this approach is that there is no means for a tutor to restrict what can be drawn to a finite number of UML-specific symbols. This makes automating their assessment more challenging. It also means that the students do not benefit from a tool that supports and enforces adherence to the underpinning methodology being taught. The ability to identify UML-specific symbols has the potential to aid assessment when, for example, searching for errors contained in the student diagram.

To summarise, a distinction can be made between the tools available for the student to draw a diagram and the format of the diagram that is submitted. Tools to support drawing enable the student to produce diagrams from a fixed set of pre-defined components. These components are derived from the underpinning pedagogic context and methodology being taught. Some systems allow the tutor to specify what these components are and the rules for how they can be connected. Conversely, others specify the components and rules in accordance with an underlying methodology. There are semantic meanings associated with the components contained within the diagram. An automated assessment system needs to consider how to represent a diagram that facilitates the identification of the diagram's constituent components.

2.4.3 Challenge 2: Model Answers and Marking Schemes

This section discusses the role that a tutor-supplied marking scheme and/or model answer plays in the automated assessment of a student submission.

In this research a *model answer* is defined to be a tutor supplied diagram that represents a solution to a problem that has been set in an assignment brief. A *marking scheme* is defined to be a prescription of how marks are to be allocated to individual diagrammatic components or to the holistic structure and format of the diagram submitted by the student. A *marking tool* is a software program that uses the model answer and marking scheme to automatically asses a student-submitted diagram. The output of the tool is formative or summative (or both) feedback.

It is difficult to produce a model answer and a marking scheme that can be used to automatically assess a diagram. This is because, for any given problem, there potentially exists several different, but equally correct or partially correct, diagram-based solutions (Soler *et al.* 2010). Consequently, for the same assignment, students could submit different but equally correct solutions. The problem is exacerbated when the student submission contains errors or extraneous components. For fixed response systems a marking scheme can be used to guide a marking tool to simply search for text in the student submission that matches that contained in a model solution. Free response diagrams are more complex and marking schemes for them need to be more detailed and go beyond symbol recognition (Tsintsifas 2002).

There are at least three approaches that existing systems have adopted in addressing this problem. The first involves the tutor producing a set of rules specifying those elements that must appear in the students' diagram and those that should not. The second involves building a database of alternative model solutions and the third involves the tutor producing a single model solution. The latter two approaches require the marking tool to contain a heuristic to search for matches between diagram components - the first requiring the heuristic to match against a list of possible correct solutions and the second requiring the heuristic to match with a single model solution.

Systems that provide alternative model solutions differ in their approach. Examples include: attempting to identify the maximum number of anticipated features that would be common to all submitted diagrams (Higgins and Bligh 2006), the tutor supplying alternative model solutions to sub-components of the model diagram (Tselonis and Sargeant 2007), undertaking a comparison with a set of alternative diagrams if a match could not be found with an initial model

solution (Suraweera and Motrovic 2002) and building a database of correct solutions as each student submission is marked (Prados *et al.* 2011).

The differences in these approaches can in part be attributed to the context and type of diagram assessment system being developed. Higgins and Bligh's (2006) system searched for a maximum number of features identified in a marking scheme. The features were those anticipated to be common to all student diagrams. They recognised that future developments for their system would need to incorporate a mechanism to support the marking of submissions from a student cohort where diagrams could be distinct, different and yet equally correct.

Tselonis and Sargeant's (2007) Gree system aimed to produce domain-specific feedback from a diagram whose internal representation was non domain-specific. They address the issue of multiple, correct diagrams through the tutor specifying alternative solutions to sub-components of the model answer. They present an example of a model solution for a UML class diagram containing 8 different sets of fully correct answers. Their marking algorithm involves representing all possible combinations of solutions in a tree-based data structure and a matching heuristic that parses each component and searches for a match with the student solution.

Suraweera and Motrovic (2002)'s Kermit system was designed to aid tutors in the teaching of Entity Relationship Modelling. The implementation of their system uses domain specific knowledge to produce a set of alternative ways of specifying similar ER structures. Their system works by comparing the student diagram with a model solution and when an exact match between entities is not found it attempts to find a match against this list of alternative, but equivalent, structures.

Prados *et al.*'s (2011) ACME system adopts a human collaborative approach. Initially there is no model solution. As the tutor marks each student submission it is stored in a database with corresponding feedback and labelled as being either correct or incorrect. Further submissions are initially compared with those stored. If a match is not found – the tutor marks it and it is stored in the database. If a match is found the feedback is retrieved and presented to the student.

One approach to creating a marking scheme is through directing a set of marking tools to search for specific features in a submission and return a mark if they find it (Tsintsifas 2002). The overall mark is the weighted sum of the marking tool responses. Higgins *et al.* (2009) developed a tool, Ariadne, to produce domain-specific marking tools in four specific areas: Logic Design, Flow Charts, Object Oriented Design and Entity Relationship Diagrams. They attempted to construct a generic marking tool that can be re-used for all future diagram domains. They concluded that tools to support marking need to be constructed each time a new diagram domain is to be assessed. They noted that this development process can be both lengthy and involved.

Thomas *et al.* (2007) have three steps in their marking method. The first is to undertake a comparison between diagrammatic components contained in the student diagram with those contained in the model diagram. The second is to calculate a similarity measure for each pair of matched components and the third is to compute a mark for the student diagram based on the similarity measures. A match is determined primarily by searching for similarities between the names of the components and their relationships.

To summarise, marking schemes and model answers are used to provide guidance when assessing a student diagram. In fixed response systems this will typically take the form of a list of items that are expected to be contained in the

student submission. Free form diagrams require more detailed configuration. The development of tools to support a given diagram domain can be both lengthy and involved. One of the challenges of producing a marking scheme is that there are several different diagrams that the student could produce all of which represent a correct or partially correct solution. One mechanism adopted by existing diagram assessment systems that address this issue is through the provision of multiple marking schemes.

2.4.4 Challenge 3: Methods for Diagram Comparison

This section discusses mechanisms for automatically comparing diagrams. This typically occurs in automated diagram assessment systems when a comparison is made between a student diagram and one supplied by the tutor as a model solution. This section discusses existing systems, the data structures that have been used to store such diagrams and the heuristics followed that undertake a comparison. This section will show that the field of model differencing has synergies with the automated assessment of student diagrams.

Automatically comparing free-from diagrams is difficult. Diagrams being compared could either match exactly, be significantly different or be ‘similar’. Defining what ‘similar’ means and producing appropriate feedback for different levels of similarity is challenging. The problem is exacerbated when dealing with imprecise diagrams that are either malformed, have features that are missing or extraneous (Smith *et al.* 2004). Imprecise diagrams frequently occur in student submissions (Smith *et al.* 2004). Furthermore, components of the diagram very often contain text-based labels. Such labels are unbounded and present the problems of synonyms, homonyms, misspellings and abbreviations (Jayal and Sheppard, 2009). They raise significant challenges relating to the fields of

artificial intelligence and natural language processing (NLP) in attempting to derive meaning from human input. However, in an educational context, imprecise student diagrams, although only partially correct, still require feedback (summative, formative or both) to be generated that is of benefit to the student.

Thomas (2004) identified three core questions that a diagram comparison system needs to address:

- how to internally represent the diagrams of both the student submission and the model diagram supplied by the tutor.
- what model or heuristic do you follow in order to undertake a comparison between the two diagrams.
- how to generate meaningful feedback (summative or formative) as a consequence of undertaking the comparison.

Approaches taken to address these questions differ from system to system. Methods to internally represent diagrams include adopting graph-based data structures, with the nodes and edges representing the entities and relationships respectively (Tselonis 2005), and those that consider the entities and relationships as separate minimal meaningful units (MMU) (Smith *el al.* 2004 and Thomas *et al.* 2005).

Methods for comparing a student diagram with one produced by a tutor can be grouped into those that compare individual diagrammatic components or those that search for patterns in the student submission. For example, in assessing ERD-diagrams, Tselonis *et al.* (2005) compared vertices on the graph generated from the model solution with those derived from the student submission, calculating a matching score for each vertex. In considering a

diagram to consist of a number of MMUs, Smith *et al.* (2010) and Thomas *et al.* (2012) enabled both a comparison of entities and the relationship that connects them, reflecting both in the resultant matching score. They report a good correlation between marks produced by their system and those generated by the academic tutors.

Examples of identifying patterns in the student submission can be found in Thomas *et al.* (2006) and Batmaz and Hinde (2006). A pattern describes the general shape of a diagram and allows the user (human or machine) to fill in details and hence specialise the diagram (Thomas *et al.* 2006). Batmaz and Hinde's (2006) semi-automated approach analyses each student-submitted database diagram and identifies sub-diagrams that are common to two or more. Semi-automation derives from their proposition that an academic tutor need only mark a sub-diagram once. Their tool, having identified the sub-diagrams in each submission, then utilises the manual marking from the academic tutor to attribute the same mark for all students whose submission contains the identified sub-diagram.

2.4.4.1 Labels

The majority of components contained in a student diagram will contain some form of text-based labelling. This is true for both the components themselves and any diagrammatic representation that attempts to link them together. One source of a potential comparison between a student diagram and a model solution is to compare the labels produced by the student to identify the individual diagrammatic components and their linkages. However, due to the free-form nature of text labels, identification of diagrammatic components within the student diagram by comparing labels contained in the student diagram with those contained in the model solution is challenging. This is because the names of entities and their relationships used by the tutor is not necessarily the same as the names used by

the student. Additionally, labels used to identify a class tend to be more succinct than the longer labels associated with relationships or use cases. The imprecise nature of labels are not elements of natural language. Consequentially they are challenging for NLP techniques to determine whether two labels are similar – have the same meaning/semantics. However, due to imprecision, ad hoc methods have to be used.

In comparing results from their Gree system, Tselonis and Sargeant (2007) attribute the difference in marks generated by their system and a human marker as being caused by insufficient label matching. The challenges posed by labels produced by students include their verboseness, the label containing defects such as misspellings, abbreviations and a different lexical structure (e.g. embedded punctuation) and students using a range of strings to indicate the same intent (Tselonis *et al*. 2005; Thomas *et al.* 2009; Higgins *et al.* 2009). Additionally, student labels potentially contain synonyms (e.g. module and unit) and homonyms (e.g. manager and clerk are hyponyms of employee).

The free-text nature of labels presents a significant challenge for systems that automate diagram assessment (Jayal and Shepperd 2009). Such challenges can be addressed through the development of a free-text similarity system (Tselonis et al. 2005; Tselonis 2008, Thomas et al. 2009 and Jayal and Shepperd 2009). Much of the intended meaning of a diagram is contained within the labels that the students produce and their absence makes a diagram difficult to understand and consequently to assess (Jayal and Shepperd, 2009).

Jayal and Shepperd (2009) report that 160 UML diagrams produced by their students contained 2013 labels with a mean of 12.58 labels per submission and

each label having a mean of 3.06 words. Their analysis indicates, even for simple diagrammatic tasks, as the number of submitted diagrams grows so too does the number of labels and synonyms of correct labels. They conclude that

"….. the problem of labels is substantial and cannot be easily avoided for the e-assessment of at least some classes of diagram." (Jayal and Shepperd, 2009)

There is a need for better algorithms to undertake a semantic analysis between labels that are contained in the student diagram and those that are contained in the model solution (Jayal and Shepperd, 2009). The technique of edit-distance alone is not adequate as the more open-ended, or subjective a question is, the more difficult the task of specifying in advance every acceptable alternative string becomes (Tselonis 2008).

Thomas *et al.* (2009) incorporated edit-distance in their technique to determine the similarity between two labels, one from a student diagram and one from a model solution. Their approach incorporates the use of Porter's (1997) stemming algorithm to identify words that are different but can be deemed to be equivalent (e.g., presenting, presented, presentation all have the same stem – "present"*)*. This is complemented with producing a domain-specific dictionary of synonyms in addition to calculating a similarity metric based on edit-distance to address misspelling in labels. They applied their technique to labels contained in 394 student diagrams. They report that students chose labels for entities from names that were contained in the assignment brief whilst labels chosen for relationships were more diverse and hence more complex to match.

2.4.4.2 Model Differencing

A comparison needs to identify features that are similar in the two diagrams and those that are erroneous or missing. There is work in the field of model differencing that is related and applicable to this issue. The field has arisen from research work undertaken to address the problem of how to maintain large-scale software systems when they are subjected to evolutionary or developmental change. The two major components of a difference tool are an algorithm that computes the difference between the two models and a mechanism to display the differences identified (Schmidt and Gloetzner 2008). Differencing analyses and compares the semantics of the models' features. Differences in their layout are considered 'irrelevant' (Ohst *et al.* 2003b).

Few algorithms and tools for computing differences between models exist (Treude *et al.* 2007). Those that do initially search for correspondences, typically by visiting each feature in the first model, conducting a search in the second and identifying that which is most similar (Chawatha *et al.* 1996, Chawatha and Garcia-Molina 1997 and Wang *et al.* 2003). Features of the model for which a match cannot be found are considered to be consequences of incremental changes made between the first and second models (Treude *et al.* 2007). The focus of feedback to the designer centres upon the collation, management and communication of a large volume of change data that represents the result of iterative incremental changes made to a system's design. Colour representations are often used (Wenzel 2008, Kelte *et al.* 2005, Ohst *et al.* 2003a and Chawathe *et al.* 1996) to highlight differences between the models being compared and as a means of managing the volume of changes being reported.

Differencing tools vary in the specific data structures adopted to represent a diagram's constituent features and the relationships between them. It is these data structures that are analysed and processed by the tool when searching for matches between two diagrams. Such representations include the adoption of structured trees (Chawathe *et al.* 1996), unstructured trees (Wang *et al.* 2003 and Chawathe and Garcia-Molina 1997) and a hybrid tree structure (Kelte *et al.* 2005 and Treude *et al.* 2007) that include 'graph-like cross references' (Kelte *et al.* 2005). Similar graph-based representations, where the nodes of the graph represent the constituent components of the diagram and the edges represent the relationships between them, were used in systems developed by Ohst *et al.* (2003b), Xing and Stroulia (2005) and Uhrig (2008).

Differencing techniques have been applied to UML models (Kelte *et al.* 2005, Egyed 2007a and Xing and Stroulia 2005). Both the SiDiff tool of Kelte *et al.* (2005) and the UMLDiff tool of Xing and Stroulia (2005) represent the diagrams in a graphical data structure with the nodes representing the entities within the diagram (e.g. the classes) and the edges of the graph representing the relationships between the nodes. They differ, however, in their requirements for how the diagrams are represented as input into the differencing tool. SDiff requires an XMI (Object Management Group 2007) description of the diagram and maps this onto its internal data structure. UML diff takes as its input two Java source code files and reverse engineers them into two separate diagrams, mapping these onto its internal data structure.

The approach adopted by Egyed (2007a) takes a different approach by modelling the impact of changes made to UML models. Impact is monitored through the establishment of a set of consistency rules that a valid UML model is required to

adhere to. An example of such a rule is that the name of a message being sent must match the name of a method in the receiving class. Incremental change that modifies either the sending message or receiving method that subsequently violates this rule would signal to the designer that an inconsistency has been introduced into the system as the design has been modified. A change in design that does not violate the rule would signal that the changes have led to a consistent design. The method adopts a semi-automated approach to resolving inconsistencies identified. It is left to the designer to decide what course of action to take once the method has identified an inconsistency. Egyed (2007a) presents 34 such consistency rules applied to 48 UML models, concluding that a tool cannot repair inconsistencies automatically but can report on all inconsistencies that arise as a consequence of a design change. The tool developed to support the specification of such rules and the identification of inconsistencies is reported in Egyed (2007b).

SiDiff adopts a two-stage pre-processing of the diagrams before the difference algorithm can be applied. The first stage translates the diagrams into an XMI format. The second stage takes these descriptions of the diagrams and maps them onto the internal data structures required for the difference algorithm to undertake a comparison. The difference algorithm operates in a bottom-up fashion starting with undertaking a comparison of the leaves within the diagrams being compared. A top-down analysis is invoked for those components for which the bottom-up approach could not produce a match.

Xing and Stroulia (2005) match components by comparing the type of the entity (e.g. comparing a class with a class), the entities name and the types of

relationships it has with other entities. They consider the name of an entity to be a safe indicator for the identification of a match arguing that

"….. it is indeed a rare phenomenon that an entity is removed and a new entity with the same name but different behaviour is added to the system."

Xing and Stroulia (2005).

As a consequence a core threshold upon which a match is determined focuses upon the name of the entities being compared. They recognise however that a new version of a system might have renamed an existing entity from a previous version. In this context their algorithm utilises the number and types of relationships between entities to identify a match. Hence, their algorithm for comparing entities consists of two components. The first is a method for comparing the names of the two entities. The second is a method for determining how similar the entities are by looking at the how they relate to other entities within the diagram.

The field of model differencing is founded in the context of incremental changes being made to large-scale systems. The development of such systems mostly takes place in teams (Kelte *et al.* 2005) and leads to the production of large-models that exist in many versions (Treude *et al.* 2007). The need for tools and utilities that calculate the differences between models arises from the need to undertake a version control of such systems (Treude *et al.* 2007, Schmidt and Gloetzner 2008 and Kelte *et al.* 2005). Consequently, some assumptions in model differencing hamper its application to analysing a student submission and providing formative feedback.

The first is the assumption that the two diagrams being compared are essentially correct. Thomasson *et al.* (2006), Smith *et al.* (2004), Thomas *et al.* (2005) and Bolloju and Leung (2006) note the presence of errors when considering diagrams produced by undergraduate students. They note that such features typify a student submission of a design diagram. Consequently, an assumption cannot be made that the two diagrams being compared are correct.

The second is that there is a strong similarity between the diagrams being compared as the two diagrams represent an evolution of the same system. This is not necessarily the case when comparing a student diagram with a tutor-supplied model solution. Higgins and Bligh (2006) note the problem of considering several distinct diagrams each one potentially representing a different but correct solution. The assumption that the two diagrams represent an evolution of the same system cannot be made.

The third relates to the feedback generated to the designer. In the field of model differencing, feedback is used to indicate where the differences lie between two diagrams. The challenge is in managing the volume of changes made and how such changes can be visualised in a manner that is useful for the design team (Wenzel 2008, Ohst *et al.* 2003a and Ohst *et al.* 2003b). In the educational context, feedback is concerned with helping a student to learn and needs to be embedded firmly in pedagogical principles. The feedback is used to develop an individual rather than manage the evolution of a system.

In conclusion, there are many similarities between the fields of automating the assessment of diagrams and model differencing. Both require a comparison of diagrams to be undertaken. An assessment system typically undertakes a

comparison between a diagram submitted by the student and a diagram that represents a model solution that has been developed by the tutor. Model differencing is concerned with maintaining large-scale software systems when they are subject to developmental or evolutionary change. Both distinguish between the data structures adopted to represent the diagram and the heuristic to undertake a component by component comparison. Despite the similarities, these different contexts make a direct porting of existing difference tools to the field of assessment challenging. However, there are many principles in model differencing that can be applied to automated assessment. These include representing the components and linkages contained in a diagram with an XML tagging structure, the use of data structures that represent a diagram in a manner that facilitates both the ease of traversal and the ease of comparison and the use of reverse engineering techniques to diagrammatically represent the design structure inherent in a diagram's implementation. The challenges of labels that adorn the components of a diagram have been highlighted. Labels are challenging primarily because it is the tutor who specifies the labels in the model solution and the student for the submission. The variability of label names are less of a concern for the field of model differencing as the diagrams being compared represent incremental evolutionary changes undertaken by the same development team and the naming of components consequently remains stable between each evolutionary iteration. The same can be argued for the case where the comparison is between a diagram submitted by the student and a diagram that represents the design components inherent in the accompanying implementation. This is because it is the student who determines the names of the components contained in both the diagram and the accompanying implementation.

2.4.5 Challenge 4: Handling of Errors Contained in Diagrams

This section discusses the types of errors that could potentially be contained within a student diagram. Existing work analysing the typical types of errors contained in work produced by students studying object oriented design and computer programming is discussed. Developments in classifying defects contained in software systems are presented. The 'inconsistent' defect classifier is highlighted as recognising that defects occur at the interface between a design and its implementation. The section concludes by proposing a blended approach to assessment automation. This consists of initially searching for typical errors that may be contained in the student diagram followed by analysing the interface between the design diagram and its associated implementation

Software systems, whether produced in an industrial or educational setting, will contain defects. Kelly and Shepard (2001) note that IBM's Orthogonal Defect Classification Scheme (ODC) for software systems contain qualifiers for defects that are "extraneous", "missing" or "incorrect". They also report upon the addition of a defect type referred to as "relationship" defining this as being "problems related to associations among procedures, data structures and objects". They propose extending the IBM ODC defect qualifiers to include an "inconsistent" qualifier to address the case where it is difficult to determine whether or not a detected defect is an issue with the design or with the code. This extension suggests that there is potential merit in investigating the provision of feedback to students based upon the consistency in structure between that specified by the design (in diagrammatic format) and that contained in the implementation (source code).

Consequently, an automated assessment system must be able to cope with errors that are contained in the student submission. Smith *et al.* (2004) and Thomas *et al.* (2005) defined imprecise diagrams to be those which contain either malformed, extraneous or missing features. They note that such features typify a student submission of a design diagram. Tselonis *et al.* (2005) noted that real data can be messy indicating that student diagrams sometimes are comprised of several disconnected graphs.

Students studying the field of object orientation find producing design diagrams challenging (Bolloju and Leung 2006, Thomasson *et al.* 2006). Misconceptions they exhibit include viewing objects as data variable or database records, restricting an object's methods exclusively for data access and assuming that a class can only be used to create a single instance (Holland *et al.* 1997)

Bolloju and Leung (2006) undertook an analysis of errors contained in UML designs produced by novice designers. They focussed upon the four UML design components of use case diagrams, use case descriptions, class diagrams and sequence diagrams. They grouped, using Lindland *et al.*'s (1994) quality framework, design errors into three different quality categories: syntactic, semantic and pragmatic. For UML class diagrams these groupings and error classifications are summarised in Figure 2.2 below.

	Error	Description
Syntactic	Missing Cardinality details of association	Association relationship has been identified but it contains no cardinality details
	Incorrect Naming of Class	An inappropriate name has been used for a class
	Incorrect Naming of Association	An inappropriate name has been used to describe the association between classes
Semantic	Wrong Cardinality	Association relationship has been identified but it contains incorrect cardinality details
	Wrong location of Attributes	Correct attributes have been identified but are attributed to the wrong class
	Wrong location of operations	Correct methods have been identified but are attributed to the wrong class
	Use of aggregation instead of association	Classes have been correctly identified as being related but an incorrect relationship has been identified (in this case it is aggregation being indicated instead of the expected association)
Pragmatic	Insufficient distinction amongst subclasses	The class hierarchy produced is not sufficiently granular to match the expected requirements
	Presence of derived or redundant attribute	Extraneous attributes contained in class

Figure 2.2: Bolloju *et al.*'s (2006) Tool for Error Classification

Furthermore, Thomasson *et al.* (2006) reported upon a study of object oriented design diagrams produced by students new to programming. Their study focused upon errors contained in UML diagrams. They produced five classifications for student errors. These are listed and described in the Figure 2.3 below:-

Error	Description
Non-referenced Classes	The student produces a design that contains a class in isolation that is not linked to any other components in the system.
References to non-existent classes	The student makes reference to a class that has not been defined in the design (e.g. an attribute of one class is defined as an instantiation of a class that does not exist)
Single Attribute Misrepresentation	This is defined as either:- The student defines one of the attributes for class A that really should be an attribute of class B. Or The student defines an attribute of class B to be an instance of a predefined language type (e.g. String) when it should be an instance of a class defined within the student's design.
Multiple Attribute Misrepresentation	The student defines multiple attributes for class A that really should be attributes of other class(es).
Multiple Object Misrepresentation	This is defined as the case where multiple objects of the same type are contained within the design when a collection (e.g. list) should be used.

Figure 2.3 : Thomasson *et al.*'s (2006) Tool for Error Classification

Thomasson *et al.* (2006) observed that the most common student error is the non-referenced-class. They hypothesised that this is due to the student recognising that the class is needed but struggles with how to integrate it with other classes contained in the design.

Whilst Bolloju and Leung (2006) usefully group design errors into syntactic, semantic and pragmatic groups they do not address issues of non-referenced classes contained in the approach adopted by Thomasson *et al.* (2006). Conversely, Thomasson *et al.* (2006) are not as detailed in their approach to

classifying errors associated with the relationship between classes. Additionally, the schemes of both Bolloju and Leung (2006) and Thomasson *et al.* (2006) do not fully address the case where a student design contains one or more extraneous classes.

Few systems that automate the assessment of diagrams consider the implication of errors contained in the student diagram propagating into the implementation. This coupled with existing work in analysing typical errors made by novice designers and undergraduate programmers leads to the possibility of developing a blended approach to assessing the student submission. Such an approach would involve two phases. The first phase would search for errors in the student design diagram informed by a bank of typical errors. The second would undertake a consistency comparison between the design diagram and its implementation. Both phases would offer the opportunity of providing formative feedback to the student and holistically could provide enhanced feedback in comparison to that generated when only one phase is undertaken in isolation.

2.4.6 Challenge 5: Feedback Generation

This section discusses the approach taken by existing systems to the generation of feedback. The distinction is made between those systems that generate feedback that is formative and those that attempt to generate a grade that is similar to that of a human marker. Semi and fully automated systems are discussed and techniques for utilising assessment to aid students in their learning are highlighted.

The automated assessment of free-form diagrams can help to facilitate a student's learning particularly when an iterative process is adopted with the

student receiving cumulative formative feedback through the repeated submission of coursework (Higgins *et al.* 2009). Existing systems differ in the approach taken to iterative feedback. Some systems attempt to support students in their learning by enabling multiple submissions of the same coursework (Soler *et al.* 2010, Suraweera and Motrovic 2002) whilst some provide a set of separate formative exercises designed to prepare the student for a summative examination (Higgins *et al,* 2009).

Iteratively receiving formative feedback enables the student to reflect upon the errors contained in the diagram and undertake further directed learning. Some systems capitalise upon this iterative approach by offering feedback that provides the solution to (some) of the errors identified (Soler *et al.* 2010 and Suraweera and Motrovic 2002). For example, Suraweera and Motrovic (2002) Kermit system divides the student errors into syntactic and semantic categories. For both categories, the system produces five levels of feedback based upon a comparison of the student diagram with a set of alternative model solutions. These levels are, correct, hint, detailed hint, all errors and solution. The first level (*correct*) indicates to the student whether or not the submission is correct. *Hint* and *Detailed Hint* both provide feedback to the student and differ in the level at which this is pitched with the former offering more generic feedback and the latter focussing upon specific details. The *all errors* level produces a list of hints on all errors detected by the system whilst a complete model solution is displayed at the *solution* level. When the student first submits an assignment the level of feedback is set at *correct*. The system supports Higgins *et al.*'s (2009) notion that such formative systems can support an iterative process to learning as Kermit increases the level of feedback given to the student with each iterative submission until the level of *Detailed Hint* is reached.

Systems that adopt a semi-automated approach to formative feedback generation can be grouped into two categories: those that require input solely from the tutor (Tselonis *et al. 2005)* and those that require input from both the tutor and the student who submitted the diagram (Hoggarth and Lockyer 1998 and Ali *et al.* 2007a). Tutor input is derived from the marking scheme and academic interpretation of the assessment information generated by the tool. For example, Tselonis *et al.*'s (2005) semi-automated system for the assessment of ER-diagrams compares a student's ERD diagram with a model solution that has been supplied by the tutor. The feedback generated is intended to provide assessment support for the academic tutor. Matches between the student diagram and the model solution are presented to the tutor in a colour-coded graphical format. The tutor analyses and interprets this output and uses it to manually provide feedback to the student.

Assessment tools that require student input prompt the student to indicate which components in their solution relate to those contained in the tutor's marking scheme. For example, Ali *et al.* (2007a), present the student with a list of symbols contained in their UML diagram and those contained in the model solution. The student is then invited to indicate which components on their diagram match with those on the model answer. The system then generates a list of feedback that describes the differences between the two diagrams. Hoggarth and Lockyer's (1998) system operates similalrly by comparing the student's diagram with a solution diagram provided by the tutor. Impreciseness in the student submission is addressed through manual intervention from the student. The student is presented with a list of components contained within their diagram and a list of components contained within the model solution. The student is required to interactively map and match the two sets of components.

Once the diagram comparison has been completed the system generates feedback based upon the differences between the two diagrams. The feedback is formative and no attempt is made to mark or summatively assess the submission. The feedback generated reports upon mismatches in symbol the types of components used, how components are connected and the addition or omission of any components when compared with the model solution.

To summarise, this section has presented an overview of existing systems and their respective approaches to the provision of feedback. The focus has been on systems that generate feedback that is formative. Approaches that encourage an iterative interaction between the assessment tool and the student have been highlighted. The differences in feedback generated between fully and semi-automated assessment systems have been identified.

2.5 Methods for Evaluating the Effectiveness of Automated Assessment Systems

This section reviews the field of the automated assessment of student diagrams and discusses how the developers of such systems have undertaken an evaluation of their results. The potential roles that both students and academic practitioners can play in evaluating the feedback generated by such systems is discussed. The applicability of these techniques to this particular research project is identified.

There are two perspectives to consider when evaluating the grading and feedback produced by an automated diagram assessment tool. The first is that of the student and the second is that of the academic tutor. The student's perspective is primarily concerned with evaluating the educational experience encountered whilst engaging with the tool. The academic tutor's primary perspective is concerned with

evaluating the accuracy of the grades and feedback that has been automatically generated.

Tuning the assessment tools is generally a form of supervised learning (Yannakoudakis *et al*. 2011) where human-generated marks are given for each sample. The submitted diagrams are divided into development and testing sets. The development set is examined during the development of the tool. The evaluation set is kept unexamined until the final evaluation of the tool. Evaluation consists of undertaking a comparison between the summative marks generated by the tool and those generated by the human marker(s). Statistical techniques used to test for significant differences or strong correlations between grades generated by the tool and those generated by the human marker(s) include calculating the Pearson correlation coefficient (Waugh *et al.* 2004 and Tselonis 2008) and Gwet's (2010) AC1 statistic (Tselonis 2008 and Thomas *et al.* 2008). The outcome of this analysis can be seen to have informed the developers on the maturity and development needs of their respective systems. For example, Tselonis *et al.* (2005) undertook a simple comparison between human and tool generated marks. They reported, for their developing system, a reasonable correlation but concluded that it was not sufficiently correlated to warrant using their systems for fully automated marking until further development had taken place. A further example can be found in Waugh *et al.* (2004). They compared summative grades generated by their tool with those generated by four independent markers. They calculated the mean and standard deviation based upon diagrams submitted by 13 volunteers. Their analysis concluded that their tool performed very similarly to the human markers.

The use of human marker(s) in the evaluation of an automated assessment tool poses the question of variability in the grades generated by the individual human markers. The method adopted by Thomas *et al.* (2007) addresses this issue. Their context was that of comparing summative marks generated by an automated assessment system with those generated by human markers. A bank of 591 student diagrams, produced in an examination, were used in the evaluation. Of these, 197 diagrams were used to support the development of their system and 394 were used to form an evaluative set. They used a group of academics to mark the exam papers (including the diagrams). Each marker marked a subset of the papers. They recognised the possibility of variability in the summative grades generated by this group and dealt with this by undertaking a further moderating marking exercise with an independent marking team. They evaluated the marks generated by the tool by comparing them with the respective moderated marks. They viewed the moderated human marks as the gold standard in which every moderated mark is absolutely correct. The automatic marker's marks are compared with the gold standard. They also compared the moderated mark with the original human marks and found the automatic marker was a better match with the moderated marks than the original human marks. In applying their automated marking system to the evaluative set, they reported that 91% of all automated grades came within 0.5 of the moderated mark but noted that this dropped to 83% when inheritance-type relationships were present in the student submission. Further refinement of their system (Thomas *et al.* 2012) improved this result to 99.7% and 97.4% for two corpora of data with the worst performance for both being only one mark difference.

Developers of automated diagram assessment tools that generate formative feedback typically evaluate their system by utilising the student body usually

through the use of a student evaluation questionnaire (Suraweera and Mitrovic 2002, Higgins and Bligh 2006, Tselonis 2008 and Higgins *et al.* 2009). Features of the formative feedback that students are typically asked to evaluate include its usefulness and the support it provided for their learning. Quantification of the extent of student engagement with the tool can be seen to have been determined by calculating the number of iterative submissions made (Higgins and Bligh 2006, Higgins *et al.* 2009 and Tselonis 2008).

Suraweera and Mitrovic (2002) evaluated their Kermit system via a questionnaire of students using a 5 point Likert (Likert, 1932) scale. The questionnaire asked about the students' experience of using the tool and the quality of feedback it generated. Furthermore, pre and post tests were used to evaluate the students' knowledge both before and after using the tool. They divided the students into two groups. One group used the Kermit system whilst the other (control group) used a tool referred to as ER tutor. ER tutor was a cut-down version of Kermit which did not provide any student feedback except for the complete solution. The pre-test result indicated that there was no significant difference in knowledge between the two groups prior to using the tool. The results of the post-test concluded that those students who had used Kermit had statistically gained more knowledge than those who had used ER tutor – the conclusion being that Kermit, with its staged approach to formative assessment, had made a positive impact upon the students' learning.

Higgins *et al.* (2009) also evaluated their tool through asking the students to undertake an evaluation questionnaire using a 5 point Likert scale. The students were asked to agree or disagree with a series of statements. The series of statements included:

- The System is easy to use.
- The feedback that I received for my submission motivated me to research further.
- I made improvements to my solutions as a result of the feedback that I received.
- The feedback was relevant to my solution.
- The diagram exercises were a good use of my time.

Similar to Suraweera and Mitrovic (2002) they also measured the knowledge of the students both before and after using the tool. They analysed results for two assignments – the second requiring a more complex diagram than the first. For both assignments they calculated the mean score for the cohort on the first submission (pre-use) and compared this to the mean score for the final submission (post-use and having received iterative formative feedback). 92% of their students used the tool. For the first assignment they report a mean number of iterative submissions as 5 per student with the cohort's initial submission averaging a mark of 49.2% and the final averaging a mark of 75.1%. For the second assignment they report a mean number of iterations per student as 9 with 50.7% and 70.1% as the cohort's average mark for the initial and final submission respectively.

Tselonis (2008) asked students to evaluate their tool by undertaking a survey. The survey asked :-

- How many times did you use the hint mechanism.
- How clearly was the feedback presented.
- How helpful was the feedback received.
- What would you suggest to make the feedback mechanism better.

They undertook a correlation analysis for the number of times feedback was asked for compared to the final mark generated and a comparison between the students' estimation of the number of feedback requests made and the actual number.

The literature discussed above identifies two evaluative perspectives. The first is that of the integrity and accuracy of the assessment and the second is the efficacy of the learning experience. Academic tutors are used in the evaluation of the former and students are used for latter. The approach to the evaluation of the research in this dissertation is to adopt both the student and the academic tutor's perspective. Formative feedback comments generated by applying a developed automated diagram assessment tool to an evaluative set of student diagrams was collated. A set of independent human marker(s) was employed to generate a further set of formative feedback comments for each student diagram. A group of academic tutors was employed to evaluate both sets of comments. The method of evaluation was through the use of a questionnaire. A statistical analysis was undertaken to test for significant differences and/or correlation between the evaluative scores generated for the two sets of comments. In order to evaluate the students' perspectives a further survey was undertaken. This took the form of presenting the student body with feedback generated by the automated assessment tool and asking the students to undertake an evaluation similar to approach adopted by Tselonis (2008) and Higgins *et al.* (2009). Details of the evaluation methodology are presented in Chapter 5.

To summarise, this section has presented a review of how developers of existing systems have evaluated their results. The role that both students and members of the academic community can play in evaluation and their contribution to this research project has been highlighted. Most systems reported in the literature deal exclusively with feedback and do not provide a summative mark and there

are very few attempts at providing evidence about the accuracy of the output from automatic marking systems. Evaluation of existing systems seem to have asked students whether the feedback was useful and have avoided the question of whether the feedback was correct.

2.6 Scoping a Framework for this Research

This section identifies the framework and direction for the remainder of this research project. This has been determined through an analysis of the literature and the subsequent discussion in the sections above.

Section 2.3 presented a review of the literature in the field of the assessment of student diagrams. The review has shown that existing systems are embryonic and deficient. The reasons for this include the free-form nature of diagrams, the possibility of many different but correct diagrammatic solutions to a given problem and the maturity of the underpinning technological and pedagogic models. Furthermore, marking and feedback are based upon the comparison of two diagrams and the existence of student errors and free-form labels alone make an accurate comparison very difficult. The literature review has not identified any attempts at using a design diagram and its accompanying implementation to produce feedback. Formative feedback at the interface between design and implementation will be of benefit as the student's learning moves from high to low levels of abstraction. This is challenging as errors contained in the student diagram may propagate into the implementation and the implementation phase itself could introduce new errors. However, one potential benefit is that the approach removes the need for a tutor-supplied model answer. Effectively, the model answer is replaced by the student's implementation of the diagram. As the student has both authored the implementation and drawn the design diagram the problems associated with naming and labelling are potentially reduced. The

absence of a tutor-supplied marking scheme will restrict the tool from producing a summative grade. However, the objective of the approach will be to provide formative feedback as the student moves between the design and implementation phases of system development. Students find this challenging, particularly when using object oriented methods. The research presented in this dissertation investigates the efficacy of such an approach. The research focus therefore is one of how to assess and generate feedback to the student based upon a comparison of a design diagram and its source code implementation.

Section 2.4.4.2 reviewed the literature in the field of model differencing. The section highlighted many obstacles facing the direct integration of existing differencing tools into the development of an automated assessment tool. There are, however, several principles within this field that are potentially applicable to the research contained in this dissertation. The approach adopted by Kelte *et al.* (2005) of using XML to describe diagram components can be applied to the student design diagram, its implementation and a tutor-supplied model solution. The need to represent the diagrammatic components in an internal data structure that facilitates a difference comparison to be made is also a principle that can be transferred as can the approach to computing differences summarised by Treude *et al.* (2007). Xing and Stroulia's (2005) technique of capturing the structure contained in source code through the adoption of a reverse engineering process can also be transferred.

However, the suitability of reverse engineering for the pedagogic context of this research project proved to be challenging and is discussed further in chapter 4. Furthermore, the exchange of documents between different tools was identified as a practical problem associated with the development of the SiDiff framework developed by Kelte *et al.* (2005). They cite this problem as being attributable to

different tools using different methods when mapping diagram elements onto XML elements.

Section 2.5 presented an overview of how existing diagram assessment systems have been evaluated. The research presented in this dissertation has been evaluated by both students and a team of expert markers. Evaluation focused upon the formative feedback comments generated by the tool. The evaluative method is reported upon further in Chapter 5.

To summarise, the aim of this research is to investigate the feasibility of applying and extending the principles and concepts of e-assessment and the assessment of diagrams to that of analysing and generating formative feedback for a design diagram and its accompanying implementation. The two main components of this research are the development of a proof of concept assessment tool and the method to evaluate the formative comments it generates. They will be informed by and build upon the principles identified and discussed in the sections above.

2.7 Summary and Conclusion

This section has addressed issues surrounding e-assessment and the automatic assessment of diagrams. An overview of the principles behind e-assessment has been discussed. The distinctions between formative vs. summative, automated vs. semi-automated and free vs. fixed response systems have been highlighted. A review of the field of the automated assessment of diagrams was presented and this was centred on the identification of five key challenges. These were the support for drawing a diagram, support for including a marking scheme, a mechanism to compare diagrams, an ability to handle errors contained in the diagram and a mechanism to provide feedback to the student. The embryonic and challenging nature of the field of automating the assessment of diagrams has been discussed. Synergies and differences between comparing diagrams for

assessment purposes and comparison techniques from the field of model differencing have been identified.

The question of what are the implications for an e-assessment system when errors contained in the student diagram propagate into the implementation has been posed. This question has been highlighted as one which contains merit for further investigation as, whilst existing work considers the automated assessment of both the design (Thomas *et al.* (2005)) and code (Blumenstein (2004)) as distinct entities, no systems have been found that address the assessment of the consistency between the two. This has been identified as the main focus for this research. In particular this research will investigate the feasibility of applying and extending the emerging techniques identified in this chapter to the context of a free form design (in diagrammatic format) and its accompanying implementation (source code). The scope will be one of fully automating the generation of formative feedback. In doing so this research needs to address the questions of how diagrams are to be represented for grading and feedback purposes, how such representations are to be analysed in order to produce feedback that is formative and how this feedback is presented to the student.

To facilitate this research an experimental tool will be developed. This tool will serve to facilitate the expansion, experimentation and evaluation of the methods and techniques discussed in this chapter. It will also serve to provide a mechanism to determine the effectiveness of these techniques as applied to this context. Their effectiveness will be evaluated by applying the tool to a bank of undergraduate student submissions and collating the formative feedback generated. A survey of both the student cohort and members of the computer science education community will be undertaken as a means of evaluating both the appropriateness and effectiveness of the collated feedback.

Chapter 3. A framework for formative assessment

The previous chapter discussed e-assessment and its application to automating the assessment of diagrams. It posed the question of how could e-assessment be applied to the case where a student submits both a design diagram and an accompanying implementation. It recognised that errors contained in the student diagram may propagate into the implementation and also the implementation itself could introduce new errors which were not originally expressed in the design diagram. Examples of free-form diagrams and their accompanying source code include UML class diagrams with their Java implementation, Entity Relationship Diagrams with their SQL implementation and SSADM data flow diagrams with their COBOL implementation.

This design/implementation context is one instance of the generic case where two artefacts represent different ways of expressing a solution to the same problem. Other examples include a requirements specification and a system design diagram, a text-based requirements specification and its mathematical representation, and an architectural design and its building specification. This chapter presents a framework that shows how related artefacts can be assessed together automatically to generate formative feedback. It discusses transforming an artefact from one domain to another as artefacts are easier to compare when they are described using a common syntax and semantics The framework focuses on the consistency between the two artefacts. The framework is illustrated by applying it to a design/implementation assessment task, using genuine, authentic coursework submissions from undergraduate Computing/Computer Science students. The research presented in this chapter has been published (Hayes 2007, Hayes *et al.* 2007a, Hayes *et al*. 2007b).

Section 3.1 of this chapter elaborates upon the educational context within which the framework has been developed. It scopes the content and context under which the students have submitted their coursework. Section 3.2 provides an example of a typical student submission and discusses its implications for the development of an automated formative assessment framework. Section 3.3 defines the generic case of comparing two artefacts. Section 3.4 presents a suite of conceptual models for an assessment framework. It concludes by presenting the model that was adopted for the remainder of this research. Section 3.5 presents an overview of reverse and forward engineering concepts in recognition that they constitute a part of the models discussed in section 3.4. Section 3.6 discusses transforming an artefact from one domain to another.

3.1 Educational Context

The motivation for this research is to automate the provision of formative feedback provided to undergraduate students studying object orientation as a component of their honours degree in Computing/Computer Science. Students are taught to use the waterfall development model (Sommerville 2007) and hence produce a design before implementation issues are considered. One benefit of this approach is that it enables the student to see the connection between the design, the program and the software development process. Liew (2005) extends this concept to include deliverables for additional stages of the requirements design, architecture design and test plans. The benefits claimed of adopting the waterfall model at the early stages of a course include the students being better prepared for modules that occur later on in the curriculum and a richer software development content in their final year dissertations.

Object-orientation is taught using an object-last approach (Hu 2004). Initially, students are introduced to fundamental imperative programming constructs. Objects are introduced subsequently with the initial focus upon object-based (class and objects) followed by object oriented constructs of inheritance, polymorphism and aggregation.

The assessment task requires the student to produce two artefacts: a design diagram and its associated implementation. It requires adherence to the software development lifecycle (Sommerville 2007) and the artefacts to be consistent. They are consistent when the design (in diagrammatic format) prescribes the structure and function contained in the implementation, and the implementation (source code) realises the design whilst adhering to its specified structure and function. Consistency is important as it enables the student to demonstrate the application of good practice and an engineering approach to the development of a software product.

The assignment deliverables from the student consist of a design (UML class diagram) and an implementation (Java source code). Design diagrams and source code implementations are examples of free-form items (as defined in Chapter 2). The learning outcome being assessed is the ability to design and implement objects. The assessment focuses upon three elements of these deliverables. These are the design diagram, the source code and the consistency between them.

3.2 An Example of a Typical Student Submission

An example of a typical second year Computing undergraduate submission is illustrated below. The intended learning outcome being assessed is the student's ability to design and implement objects. The example contains two related artefacts:

1) the design diagram submitted by the student (Figure 3.1)

2) the accompanying implementation submitted by the student (Figure 3.2)

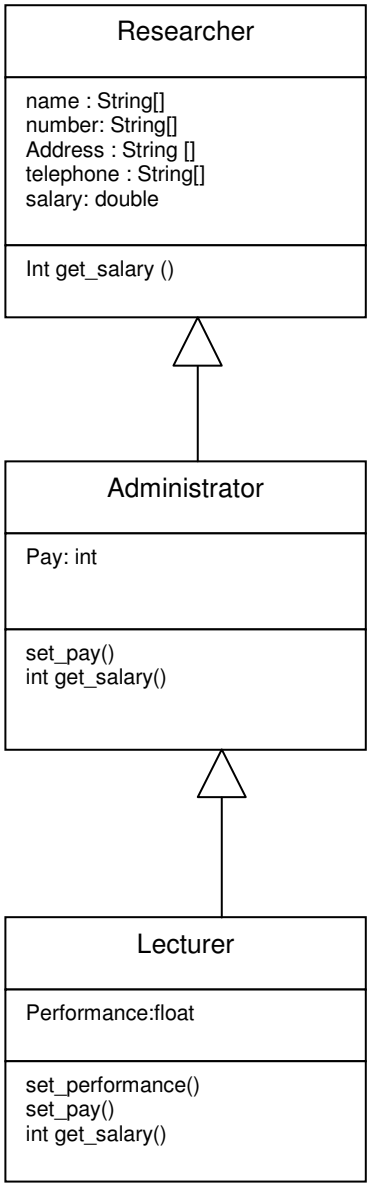


Figure 3.1 Design Diagram As Submitted by the Student

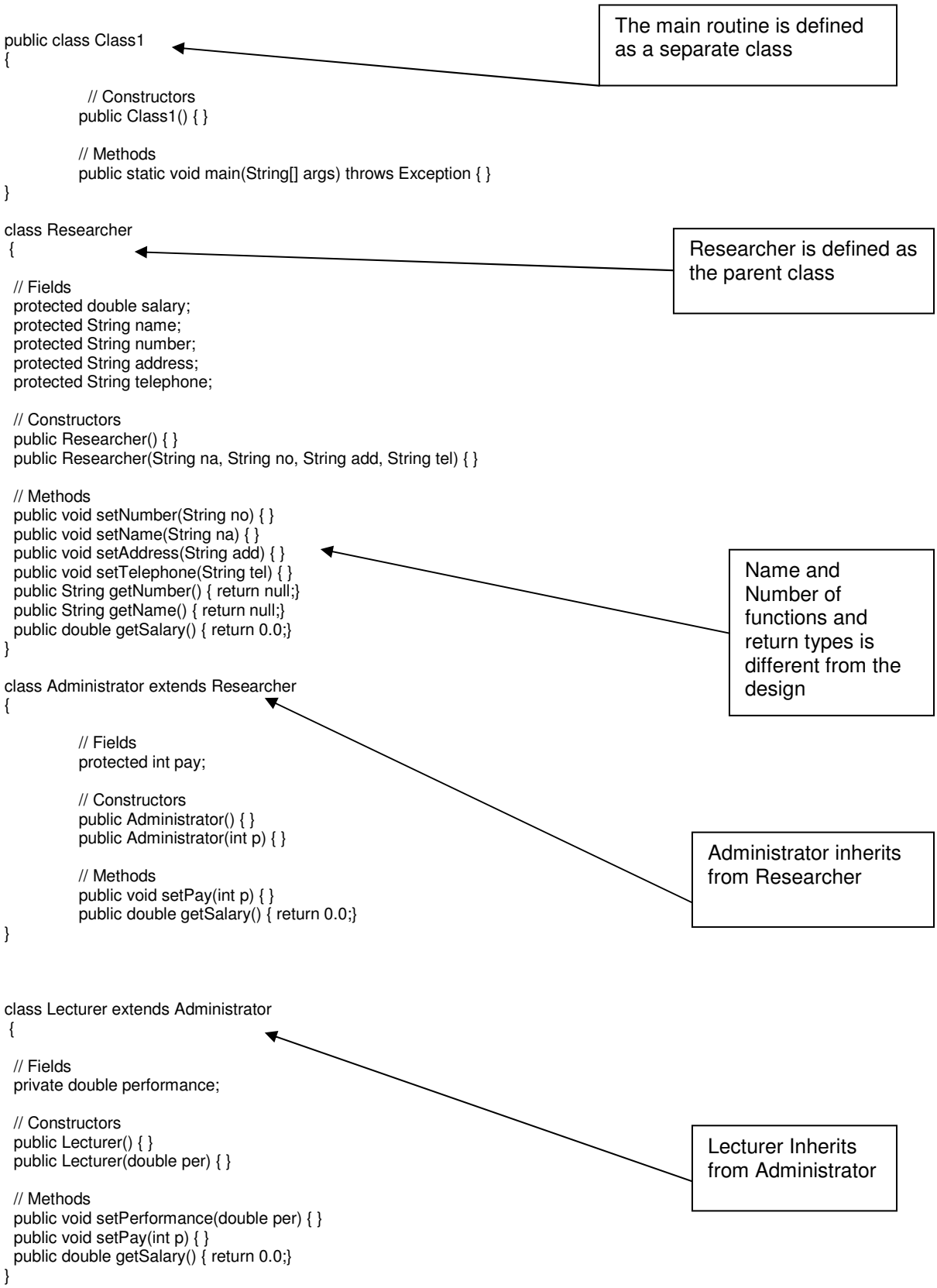


Figure 3.2: An extract of the implementation as submitted by the student

Comparing the design diagram in Figure 3.1 with the source code in Figure 3.2 raises a number of issues. There is a reasonable level of consistency between the two artefacts. The number, name and relationships between the classes match those in the design. There are some discrepancies between the number and name of some of the methods and attributes of the classes identified. This will not always be the case for other student submissions.

Figure 3.3 contains a third artefact, the expected design diagram taken from a tutor-supplied mark sheet.

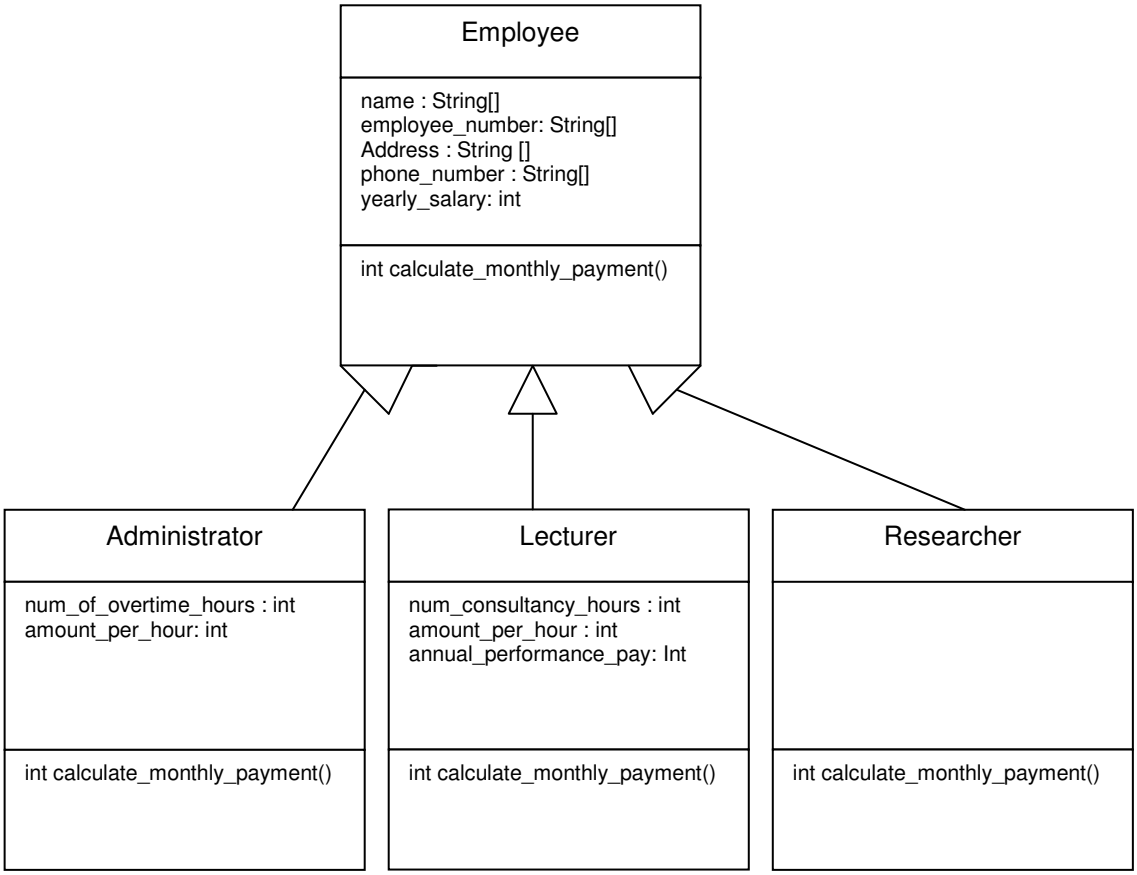


Figure 3.3 Expected Design Taken from a Tutor Supplied Mark Sheet

Comparing the tutor's design diagram (Figure 3.3) with the student's (Figure 3.1) raises further issues. The student has correctly identified three of the four required classes in addition to the inheritance relationship, although the hierarchy itself is not what was expected.

All three artefacts represent different views of a solution to the problem contained in the assignment brief. In order for this to take place there needs to be a mechanism within the framework that:

1) identifies the features within the artefacts that are being compared (in this example it is the classes and their relationships);
2) traverses, analyses and compares structures and features contained within the artefacts.
3) specifies the feedback to be generated when consistency and/or inconsistencies are identified;

The problem is complex because the student diagram may contain errors. Some of these errors will propagate into the implementation. The implementation itself could introduce new errors. A student may produce a diagram that is topologically correct but uses symbols and notation different from that expected. This poses questions such as whether the student understand the relationship and just used the incorrect linkage notation or have they misunderstood what the relationship means? The diagrams could be submitted partially complete. Sub-parts of the diagram could be correct and others not. A system that automatically generates formative feedback will need to address all these issues. It will need to go beyond the mechanism of component and symbol recognition as there is a need to consider and contextualise the semantics that each symbol represents.

In summary, this section has presented an example of three related artefacts : a design diagram from a tutor, one from a student and a student-produced accompanying implementation. It has highlighted that the same construct, in this case an assignment brief, can lead to many different representations of a

solution. It has identified some of the challenges that multiple artefacts present to automatically generating formative feedback.

3.3 Comparing Artefacts – The Generic Case

The design/implementation context is one instance of the generic case where two artefacts provide different views of the same referent. The purpose of this section is to introduce definitions for the generic case of artefacts and the concepts that arise in their comparison.

At the top level, a *construct* is a fundamental component from which several distinct descriptions can be produced. For example, an assignment brief is a construct from which a student describes a solution using a variety of abstractions and notations.

An *artefact* is a description of some construct. For instance, a UML class diagram and its Java implementation are both artefacts that partially describe a running computer system (construct). Artefacts are *well formed* if they conform to a defined set of rules, for example, the code is a runnable Java program and the diagram conforms to the UML class diagramming rules.

Figure 3.4 illustrates the relationship between two artefacts and a construct.

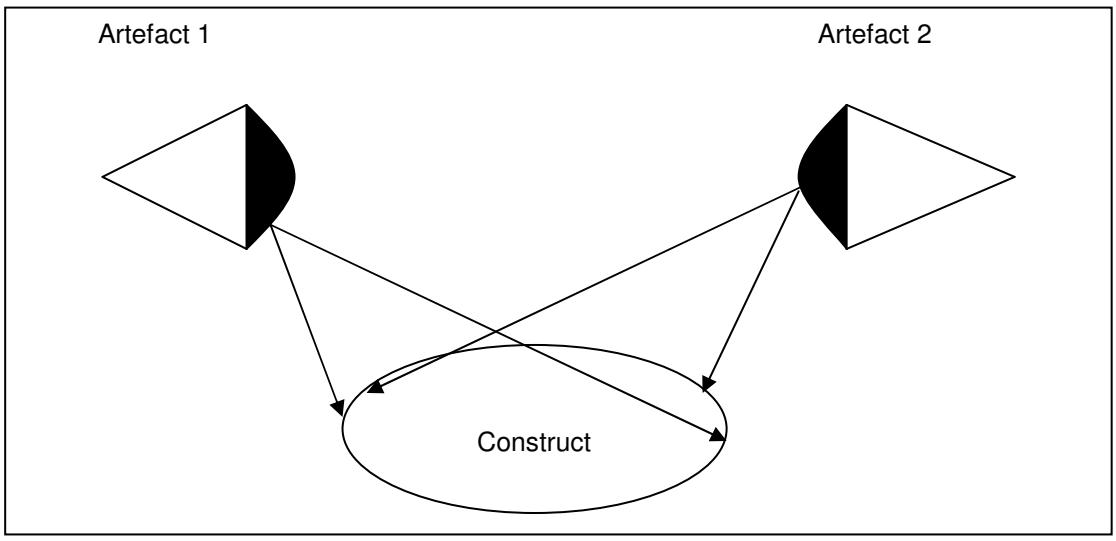


Figure 3.4: Diagram to show how two artefacts view the same construct from differing perspectives

The *features* of an artefact are the ideas, abstractions and constructions contained in its description. Features in a diagram are represented as boxes, lines, directed arrows and labels. For instance, the features of a UML class diagram are the classes and their relationships. Features in source code are identified using language-specific key words. For instance, the features contained in a java implementation are identified by the keywords *class, extends* and *new*.

A set of artefacts is *consistent* when all of their features agree i.e. for each feature in one artefact there is a one-to-one mapping onto a feature in the other. For instance, a UML class diagram and a fragment of Java source code contain the same set of classes and the same set of relationships.

A set of artefacts is *partially consistent* if some but not all of their features agree.

A set of artefacts is *completely inconsistent* if none of their features agree.

For partially consistent artefact sets, the *consistent features* of an artefact are the features implied by both artefacts and the *superfluous features* of an artefact are the features of that artefact alone.

The *consistency differences* of the artefact set is the union of the superfluous features and the *consistency similarities* is the union of the consistent features.

(Later in the dissertation it will be shown that the consistency similarities and differences between a design and implementation form a good basis for generating formative feedback).

An example of applying these definitions is presented below. The artefacts are represented by two diagrams: one produced by the tutor (TD) and one by the student (SD). Both describe their features using the UML diagram type, syntax and semantics. Figure 3.5 below illustrates the example. The construct itself is represented by the assignment brief.

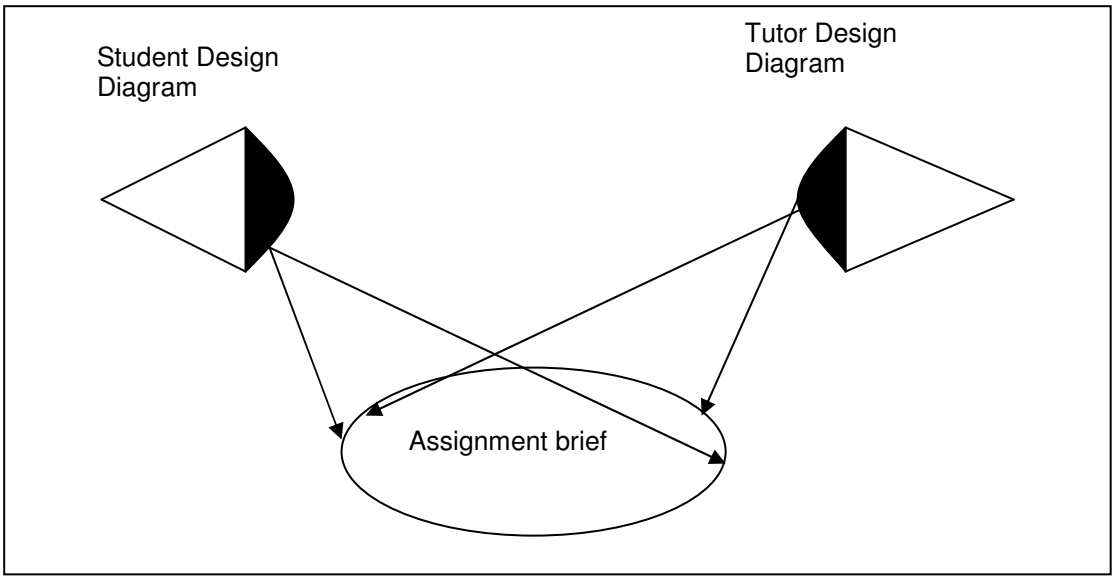


Figure 3.5: Diagram to illustrate the concepts of constructs and multiple artefacts applied to the case where a comparison is being made between a student design diagram and a design diagram produced by the tutor.

The assignment brief contains many features that a tutor expects to appear in a student solution. The two artefacts, SD and TD, represent two views of the requirements of the assignment brief. The consistent features are those contained in both SD and TD. The superfluous features of TD (those features not appearing in SD) represent omissions from the student submission and those superfluous features in SD that do not appear in TD are erroneous features. These three distinct areas are illustrated in Figure 3.6. For the feedback to be holistic, a comparison of TD with SD needs to report upon the features contained in all three.

Both the consistent and superfluous features can be analysed to provide formative feedback. Feedback upon the consistent features reinforces the positive aspects of the submission whilst the two sets of superfluous features can be used to inform the student where there are perceived problems with what has been submitted. In this example the problems are associated with inconsistencies between the tutor's model solution and the student submission.

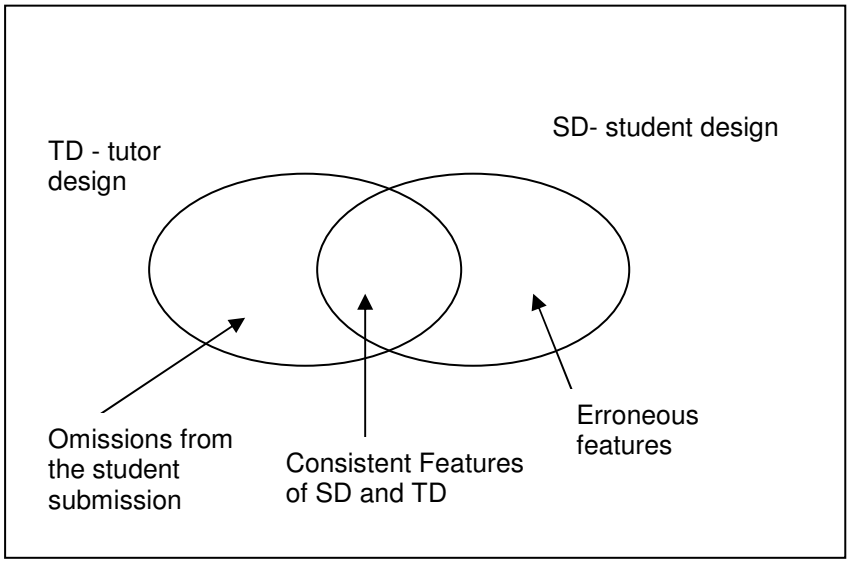


Figure 3.6: A diagram depicting the relationships contained within the student diagram and that supplied by the tutor

3.4 Models for the Assessment Framework

This section presents an overview of several high-level techniques for how a framework could analyse and feed back upon the student submission. No attempt is made, at this stage, to consider the internal operational detail of the techniques presented. The focus, instead, is to consider the inputs that such techniques might require and to identify and discuss the operational challenges that each technique presents. The relative merit of each technique is presented. The section concludes with the identification of the technique that was adopted for the implementation phase of the remainder of this research.

The context of the approach taken is illustrated in Figure 3.7 below. The student submission consists of two separate artefacts: a design diagram and an implementation.

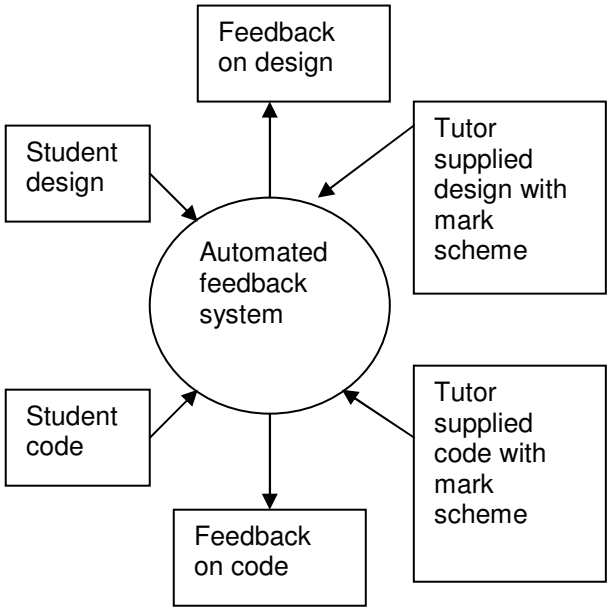


Figure 3.7: Initial Context of an Automated Feedback System

If the two artefacts were treated as disjunctive, non-related deliverables it would be possible to divide the automated feedback system into two distinct components, one focusing on the design and one on the implementation (Figure 3.8).

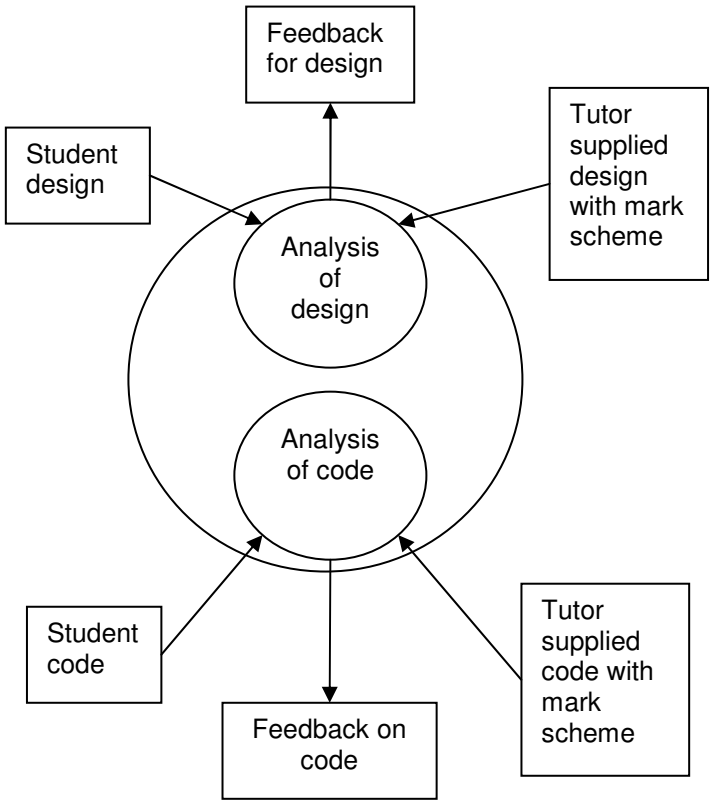
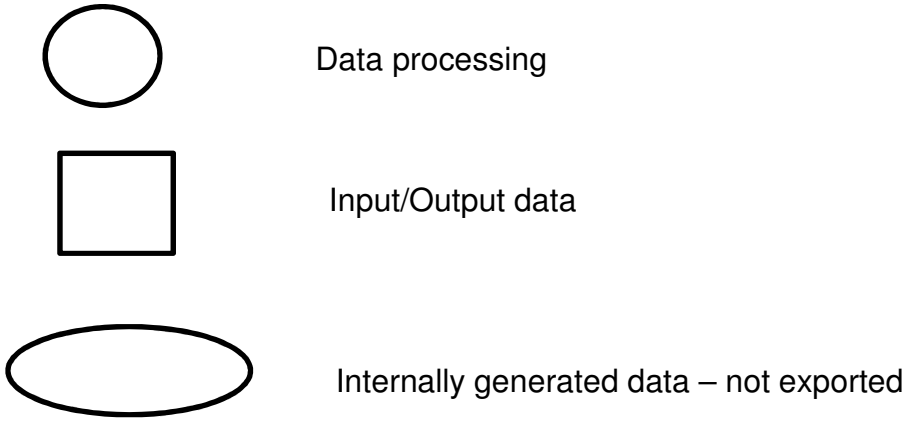


Figure 3.8: A system that marks the design and the code disjunctively

However, such an approach does not lend itself to focusing upon the interface between the code and the design. When considering feedback for consistency there needs to be a mechanism to link the structure of the student code to that of the accompanying design. This applies to the cases when the design and implementation are submitted together, the submission date is different for each deliverable (to allow for feedback to be given on the design before the student embarks upon the implementation) or when the design and implementation assignments are contained within two separately delivered modules (integrative assignment). In all cases, the student is required to produce more than just a design and a separate implementation. The two artefacts need to be consistent as together they represent a solution to the same problem.

3.4.1 Inferred Structures and Generating Feedback

There are several models that emerge for the framework. This section discusses three. Each offers a different perspective upon the student submission and consequently a different input into feedback generation. The models are illustrated (Figures 3.9 to 3.14 inclusive) using the following notation:



The first method requires, using an appropriate tool, forward engineering the student's diagram to produce an idealised structure for the submitted code (Figure 3.9). In this context, forward engineering aids the comparison by

identifying the features contained in the diagram artefact and representing them using the syntax and notation of the code artefact. A comparison could then take place between the student's code and that inferred from the design (Figure 3.10). This is referred to as a code-centric method. The superfluous and consistent features identified in the comparison could be used to generate feedback.

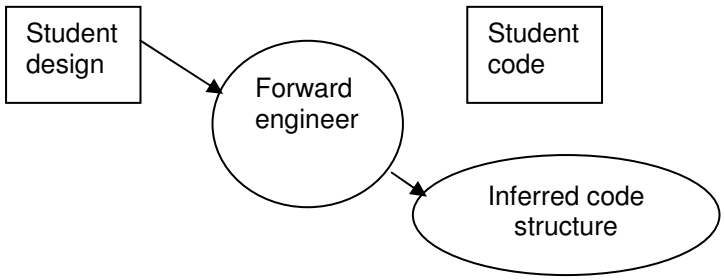


Figure 3.9: Forward Engineer the Design to produce the inferred code structure

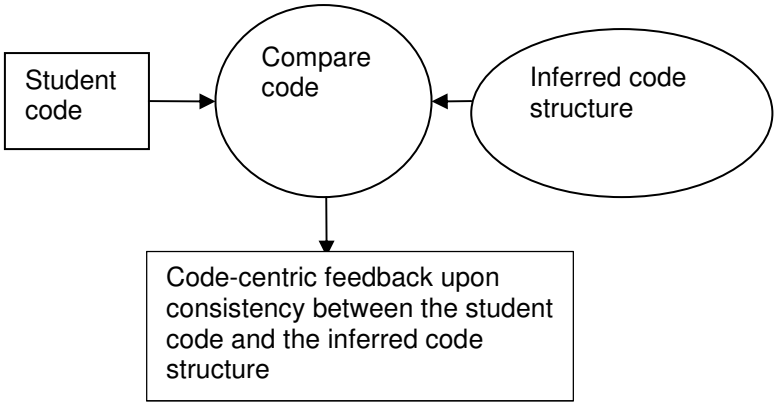


Figure 3.10: A model comparing the student code with the inferred code structure.

Similarly, the second method requires, with an appropriate tool, reverse engineering the student code (Figure 3.11) to produce an idealised structure for the design diagram. A comparison could then take place between the student's design and that inferred from the code (Figure 3.12). This is referred to as a design-centric method.

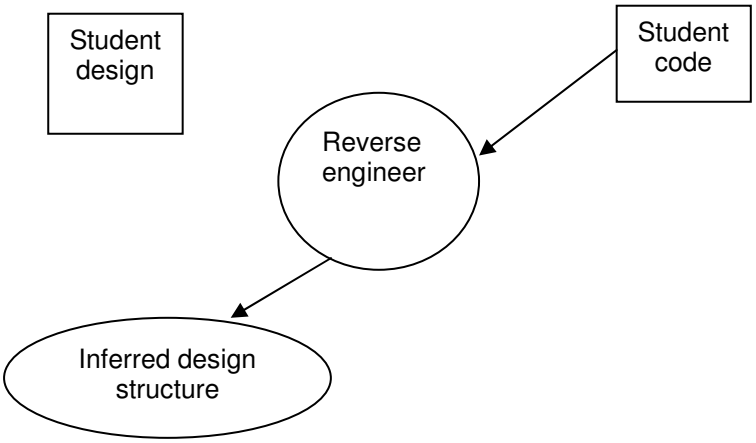


Figure 3.11: Reverse engineer the code to produce the inferred design structure

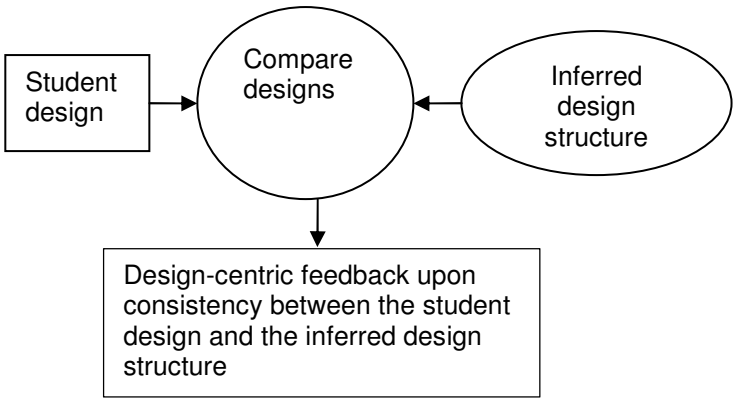


Figure 3.12: A method that focuses upon comparing the student design with the inferred design structure

An implementation of the framework could adopt either one of the design or code-centric methods. Feedback would be generated from the consistent and superfluous features identified. It is possible to imagine a tool that would implement both methods. Ideally, the results from the code and design-centric approaches would be the same. This third method is one that would triangulate between the outputs of the first and the second (Figure 3.13).

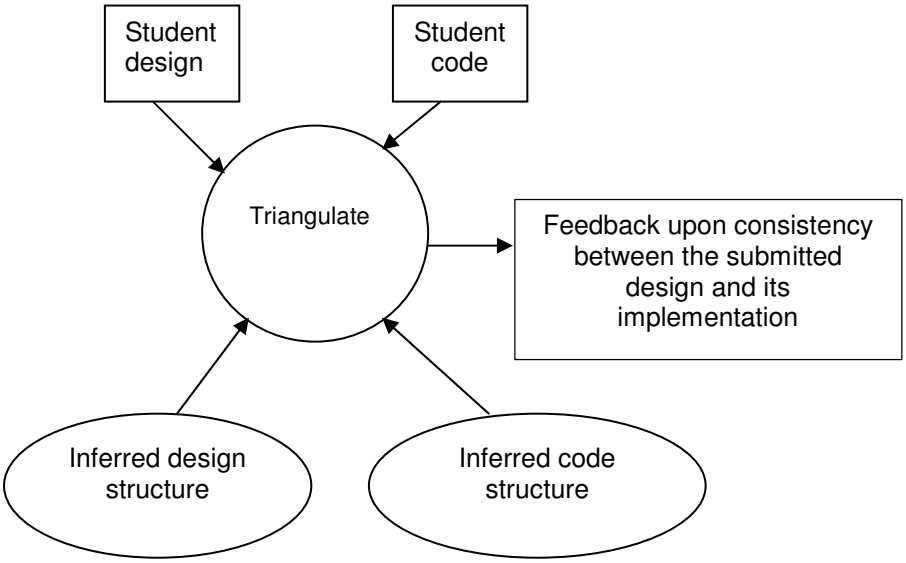


Figure 3.13: Triangulate the Assessment of the student submission with both the inferred code structure and inferred design structure

Triangulation offers the benefit of confirming that errors in the student submission have been identified by both the design and code-centric approaches. It also offers the potential of reporting upon any errors that may have been missed by one method but identified in the other.

3.4.2 Framework Support for Tutor Input

The methods presented in section 3.4.1 focused exclusively upon consistency in the student submission. However, a tutor may wish to provide additional feedback to the student. For example, the tutor might wish to feedback upon the quality of the design, its accompanying implementation or both in addition to those issues surrounding consistency. In this case, the tutor would need to specify the specific design or implementation features to be looked for and fed back upon. This enhancement, applied to the design-centric method, is illustrated in Figure 3.14.

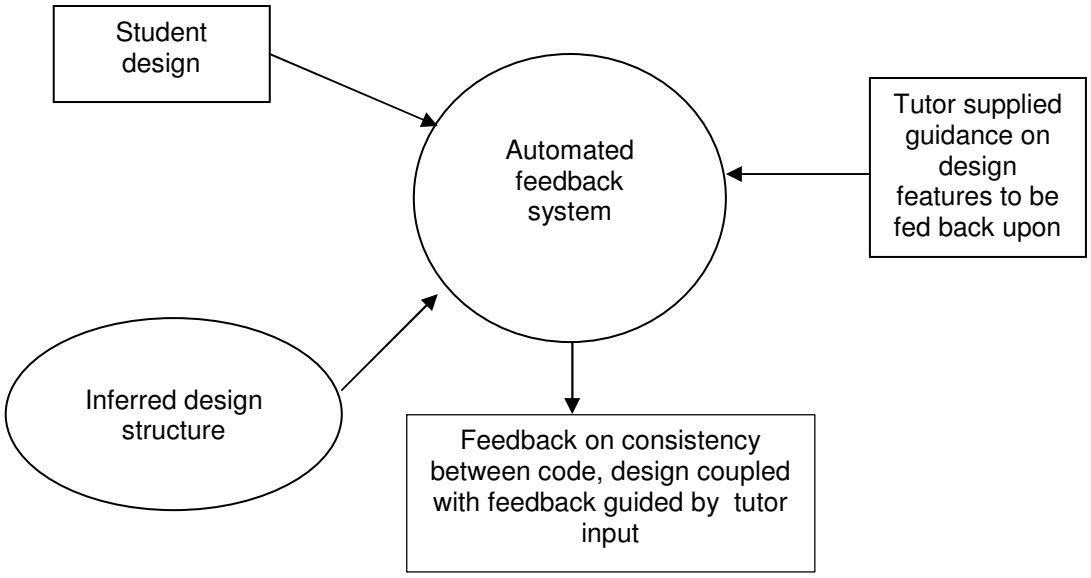


Figure 3.14: A model that generates feedback on consistency between the student submitted design and implementation in addition to feedback upon the design features requested by the tutor

3.4.3 The Model Adopted for the Remainder of this Research

To further this research a proof-of-concept tool was developed. This tool served to facilitate the expansion, experimentation and evaluation of the methods and techniques discussed above. The tool adopted the method illustrated in Figure 3.14 above. Specifically, this consists of:

1. searching the student design diagram in isolation for errors typically made by novice designers and generating feedback on their presence/absence. This is treated as default tutor guidance for the tool as discussed above;
2. reverse engineering the student code to produce an inferred design diagram;
3. comparing the inferred diagram with that submitted by the student and generating feedback upon their consistency.

The design-centric approach was adopted as it requires a comparison of two diagrams (one submitted by the student and one inferred from the source code). This presented the opportunity to investigate how existing diagram assessment techniques could be extended and applied to the multiple artefact context.

Additionally, research undertaken identified the existence of many round-trip engineering tools that offered the potential of extracting a design structure from the submitted source code (reverse engineering). Tutor supplied guidance took the form of specifying that the tool search for typical errors found in novice student design diagrams as articulated in Bollujo and Leung (2006) and Thomasson et al (2006). This is discussed further in Chapter 4 where an implementation of the framework is presented.

3.5 Reverse Engineering and Support for Feedback

The previous section signalled the intention to develop an automated feedback tool that followed the design-centric model as illustrated in Figure 3.14. This requires reverse engineering the submitted source code to produce an inferred design. This is a significant challenge as the student submission potentially contains errors and/or erroneous data. How reverse engineering techniques resolve such ambiguities in the context of assessing the student submission is an issue that needed to be addressed. Therefore, this section presents a definition of reverse engineering and highlights how it can be used to infer a design from source code and discusses how feedback can be generated by comparing the inferred design with the original design diagram.

Chikofsky and Cross (1990) define reverse engineering as the process of analysing a subject system to:

a) identify the system's components and their interrelationships and
b) create representations of the system in another form or at a higher level of abstraction.

Tilley (2000) indicates that there are three canonical activities that characterise reverse engineering. These are data gathering, knowledge management and information exploration. Data gathering is concerned with parsing (static analysis) or running (dynamic analysis) the source code that is being reverse engineered. Knowledge management is concerned with creating domain models that represent and reason about the constructs and elements contained within the source code. Information exploration involves navigating, traversing and analysing the models produced. Tilley (2000) argues that it is information exploration that increases the understanding of the source code.

A review of the literature reveals that there are many examples of how reverse and forward engineering can be used to infer structures between design diagrams and source code. Examples include the engineering of Java byte-code to UML diagrams (Cooper *et al*. 2004), OMT diagrams to C++ source code (Antoniol et al 2000), Java source code to UML diagrams (Alphonce and Martin 2005) and C++ source code to UML diagrams (Matzko *et al.* 2002). Examples of producing a diagram from the source code by extracting static relationships can be found in Cooper *et al*. (2004) and Matzko *et al*. (2002).

The literature also reveals examples of feedback being generated by comparing a given and inferred design. Examples include feedback to professional developers (Cooper *et al.* 2004) and feedback to novices in a pedagogic context (Alphonce and Ventura, 2005). Cooper *et al*. (2004) automatically compared

their diagrams with those generated by a CASE tools to identify the differences between the design and implementation. A limitation of their technique is that they did not attempt to resolve automatically the differences identified. It required human intervention through structured code review to undertake any resolution.

Alphonce and Ventura (2003) presented a tool that enabled a user to draw UML class diagrams from which Java source code was generated. It also generated UML diagrams from a given Java source. Alphonce and Martin (2005) made it compatible with Eclipse (Eclipse Foundation 2006). They used it in the teaching of introductory object-oriented programming and claimed benefits for both the tutor and the student. The benefit for the tutor was in obtaining an accurate design diagram from the student's submitted source code. However, they required the results to be analysed manually. They argued that doing this would "…make it significantly more likely that a student's design grade will actually reflect the quality of their design." (Alphonce and Martin 2005). The benefit for the student was to be able to traverse between the source code and design views of their submission.

Consequently, Alphonce and Martin (2005) recognised the need to provide feedback to the student on issues at the design-code interface. Their tool supported the ability for the student to be able to traverse between the source code and design views of the submission. However, the approach is not automated and the focus of the feedback is to enable the student to iterate through the design and code views of their development.

This section has presented a brief overview of examples taken from the literature of the application of reverse and forward engineering techniques. Whilst many examples exist, few focus on the goal of producing formative

feedback in a pedagogic context. There remains the need to investigate whether or not existing case tools are robust and appropriate enough to handle errors and erroneous data in the student submission. The problem of how to generate formative feedback from a comparison between the inferred structures also remains to be addressed. Chapter 4 will elaborate further on these issues and will discuss the findings of applying Borland's JBuilder Enterprise (a commercially available round-trip engineering tool) to this pedagogic context.

3.6 Multiple Artefacts and Transformations

This section discusses the issue of transforming an artefact from one domain to another. We wish to do this because it is easier to compare the artefacts' features when they are described using a common syntax and semantics. Examples include

- Forward engineering an artefact from the UML class diagram domain to produce inferred source code.
- Reverse engineering an artefact from the Java source code domain to produce an inferred diagram.
- Transforming both the design and inferred diagrams into a domain required by a tool that will automatically compare them.

Figure 3.15 provides an illustrative example of where two artefacts describe their respective features using notations with different syntax and semantics.

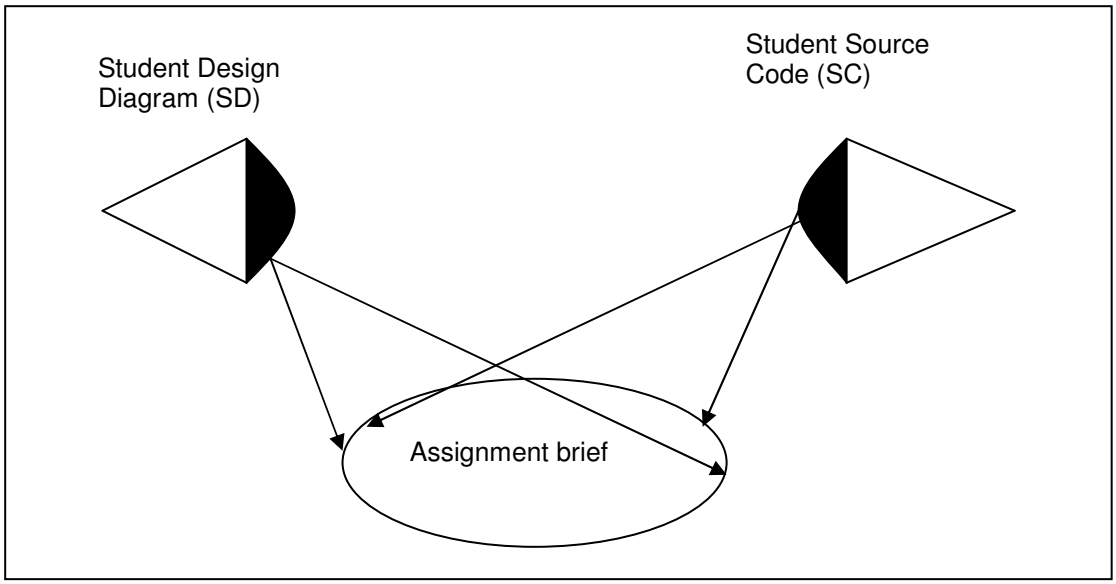


Figure 3.15: Diagram to illustrate the concepts of components and multiple artefacts (as illustrated in Figure 3.4) applied to the case where a comparison is being made between a student design diagram and student submitted code

When the two artefacts being compared originate from different domains a direct comparison cannot be made because they have different forms of syntax and semantics to represent their respective features e.g. a student diagram (SD) expressing its features using the syntax and semantics of the UML class diagram, and its implementation (SC) using the syntax and semantics of the Java programming language.

It would be possible to perform a comparison between SD and SC if either one could be transformed into the domain of the other. There are potentially many possible ways of doing this. The sections above have illustrated how forward and reverse engineering techniques could be used to perform the transformation. The example below discusses the issues surrounding a transformation from the diagram domain D to the Java domain J. The underpinning pedagogic context is that of a student exploring the connection between the design, the code and the software development process. Feedback upon how these artefacts compare will aid the student in his/her learning. The

artefact is described in the syntax and semantics of the UML class diagram. The transformation produces an artefact described in the Java programming language syntax and semantics. This requires applying a transformation mapping, f, to the features contained in the design diagram to produce corresponding features contained in the Java program domain. This is illustrated in Figure 3.16 below:

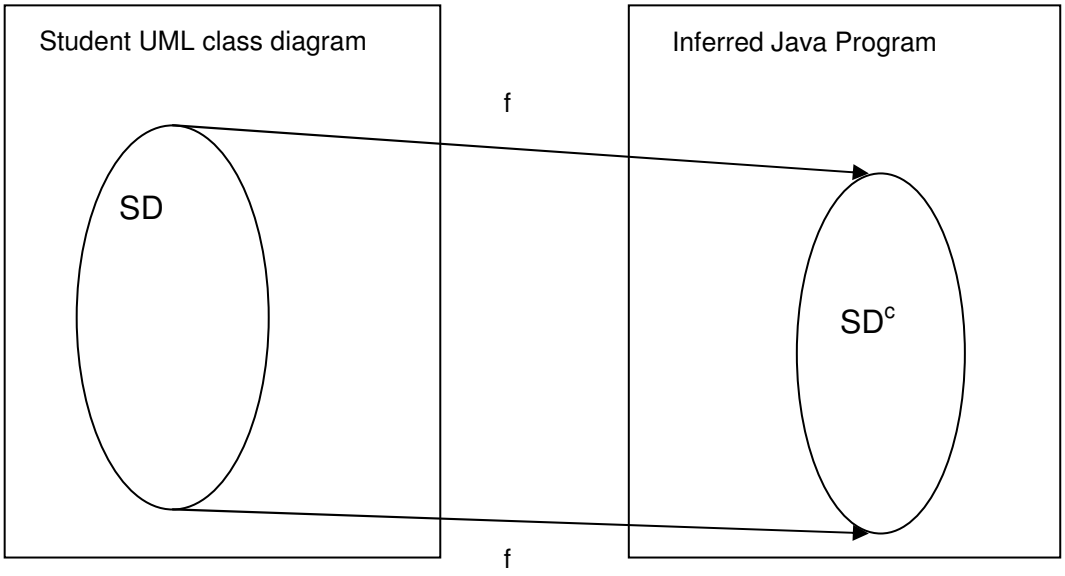


Figure 3.16: Diagram to illustrate mapping of a student diagram into the program co-domain

We can define f to be a mapping that takes a student diagram (described using the UML class diagramming, syntax and semantics) to produce an inferred student program (using the java syntax and semantics). The mapping f takes each feature in SD and for each creates one new feature in the image set SD^c . We assume that f is one-to-one, onto and that no additional features are added. Having transformed the student diagram into domain J we can then undertake a meaningful comparison as outlined above as both artefacts now describe features using the same syntax and semantics.

In an ideal world, transformation f would not lose or add anything i.e. no extra features are introduced and none are lost during the transformation. In reality the

transformation function, f, may lose some of the features contained in SD or may add some extraneous data. This is illustrated in Figure 3.17 below.

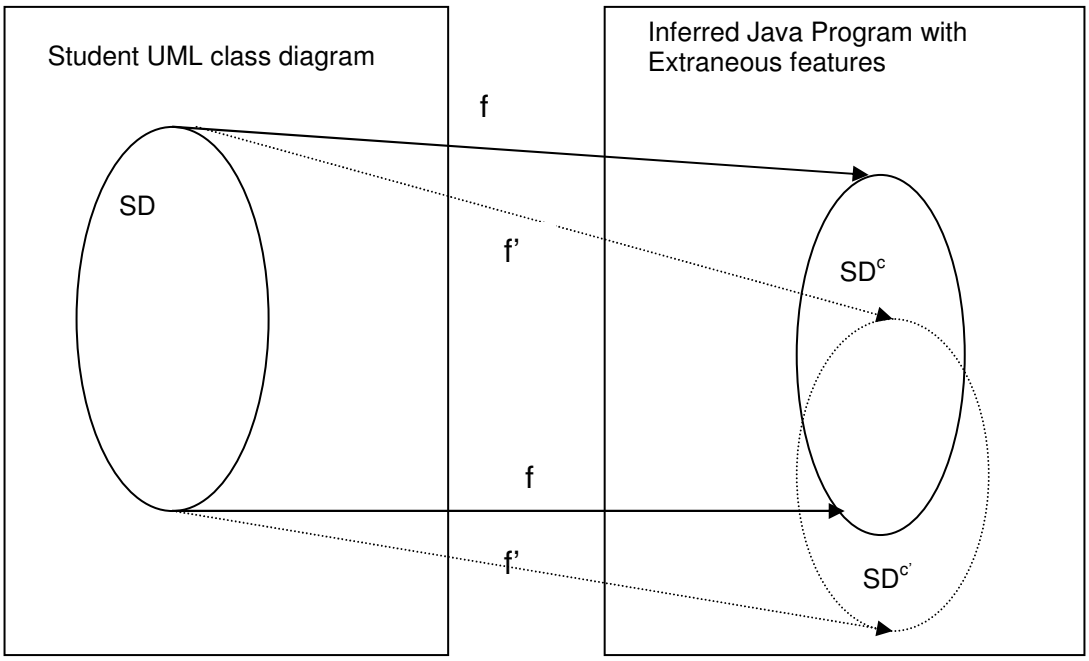


Figure 3.17 : the image set of a domain transformation f (generating no errors) and f' (generating additional errors)

When comparing the inferred code ($SD^{c'}$) with the code submitted by the student there is a need to distinguish between erroneous features contained in the student's original submission and those that may have been generated by the transformation process.

3.6.1 Transforming artefacts into the domain of an automated framework

This section discusses how the framework can compare artefacts independently of the representations used to describe them. The two representations could each be mapped to a third representation. For example, it might be possible to map both a design diagram (represented using the UML class diagram syntax and semantics) with an inferred diagram (using the same syntax and semantics) into XML, and this is what has been pursued in this research. The required

transformations can be undertaken in the same fashion as discussed above. Figure 3.18 below illustrates this approach.

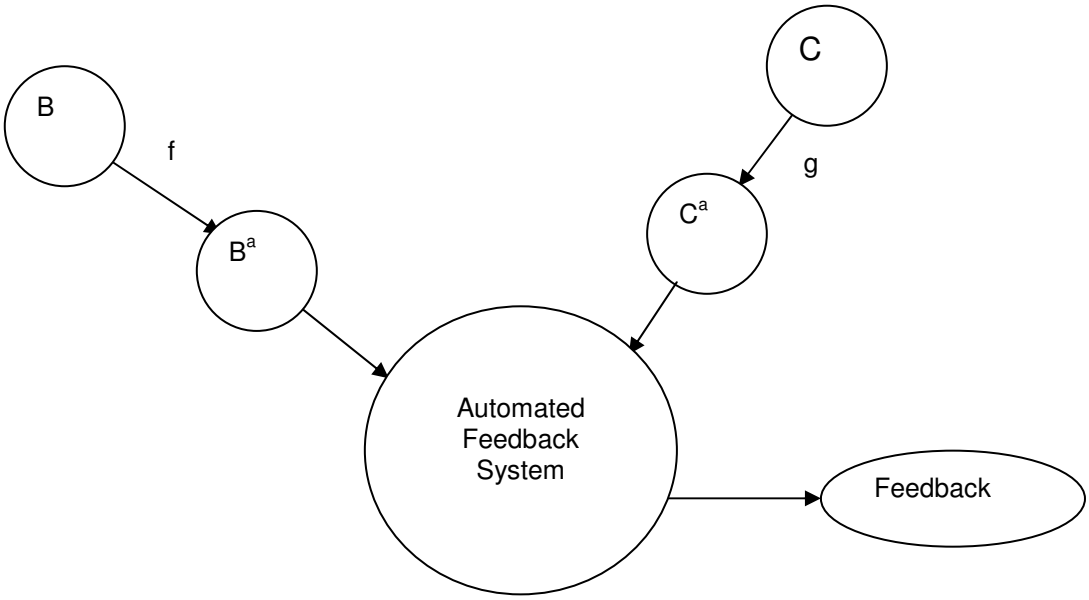


Figure 3.18 : A diagram to illustrate how two linked artefacts could be compared by transforming them into the domain of the framework.

Artefacts B and C are being automatically compared. This could represent, for example, comparing a design diagram with an inferred diagram, source code with inferred source code or an architectural design with a building specification. Each of these three artefact sets describe their features using different representations. Ideally, an implementation of the framework would be independent to any domain-specific representation. The mappings, f and g, transform the features contained in artefacts B and C respectively into the domain required by the automated framework.

Transforming artefacts in this fashion leads to the development of an automated framework that is not dependent upon the syntax and semantics of the artefacts being compared. This offers the advantage of a single implementation of the framework being able to process artefacts from a wide-range of domains.

3.7 Summary

This chapter introduced the concept of multiple artefacts. Definitions for a construct, an artefact and its features were presented. Comparing artefacts identifies consistent and superfluous features from which formative feedback can be generated. These concepts were applied to the design/implementation context. The student submission has been scoped to that of a UML class diagram and an accompanying Java source code implementation. Several models for an assessment framework were discussed. Code-centric, design-centric and triangulation models were presented. Reverse and forward engineering techniques were proposed as a means of identifying one artefact's features and transforming them into the domain of another. The issue of a transformation creating errors in the resultant artefact has been identified and discussed. The advantages of artefacts describing their features using a tool-specified language and syntax has been highlighted. The design-centric model, blended with searching for typical undergraduate diagram errors, has been signalled as the basis for the development of a proof-of-concept development tool. The development of this tool is discussed in Chapter 4. Chapter 4 will report upon an experiment undertaken to apply reverse/forward engineering tools to this pedagogic context. It presents a heuristic developed for the comparison of two artefacts and identifies how this can be used to automatically generate formative feedback.

Chapter 4. Development of the Formative Assessment Tool

The previous chapter presented a framework for the automatic generation of formative feedback. It introduced the concept of the student submission consisting of multiple artefacts. This chapter presents an application of the framework. A formative assessment tool is presented.

The aim of developing the tool was to facilitate the expansion, application, experimentation and evaluation of the multiple artefact concepts discussed in the previous chapter. The tool automatically generates formative feedback based upon an analysis of the student submission.

The submission serves as an illustrative example of the multiple artefact context. It consists of two artefacts – a UML design diagram and an accompanying Java implementation. This chapter presents the mechanism adopted to describe the features contained in both artefacts and the heuristic developed to analyse these descriptions.

Section 4.1 provides a high level overview of the developed tool. Section 4.2 discusses the application of forward and reverse engineering techniques to the submitted artefacts. Section 4.3 defines the grammar structure developed to describe the artefacts' features. Section 4.4 presents the heuristic developed to compare the artefacts and generate formative feedback. Section 4.5 discusses the mechanism adopted to search for typical errors made by undergraduate students. Section 4.6 presents an example of two artefacts submitted by a student and the feedback generated by the tool.

4.1 High Level System View

This section provides an overview of the developed assessment tool and the context under which it operates. The educational goal is to provide formative feedback as the student's learning moves from high to low levels of abstraction. There is no restriction placed upon the number of times that a student can submit their work to the tool: the rationale being that it provides formative support to aid learning rather than a summative judgement on what has been learnt.

Figure 4.1 contains a flow chart that illustrates the main system components. The process takes the student submission as input, analyses it and produces formative feedback. The input consists of two artefacts: a student submitted diagram and its accompanying implementation. Initially, the two artefacts are transformed into a format that the tool can recognise. This translation is referred to as tagging and is currently undertaken manually. The split between the manual and automated parts of the process are considered further in section 4.2. The result of tagging a student design diagram is referred to as a tagged diagram. Similarly, the result of tagging an implementation is referred to as a tagged implementation. Analysis of the artefacts takes place in two stages. The first compares them and the second looks for design errors contained in the tagged diagram. Formative feedback is produced at the end of each stage. The remaining sections in this chapter provide further details on these system components.

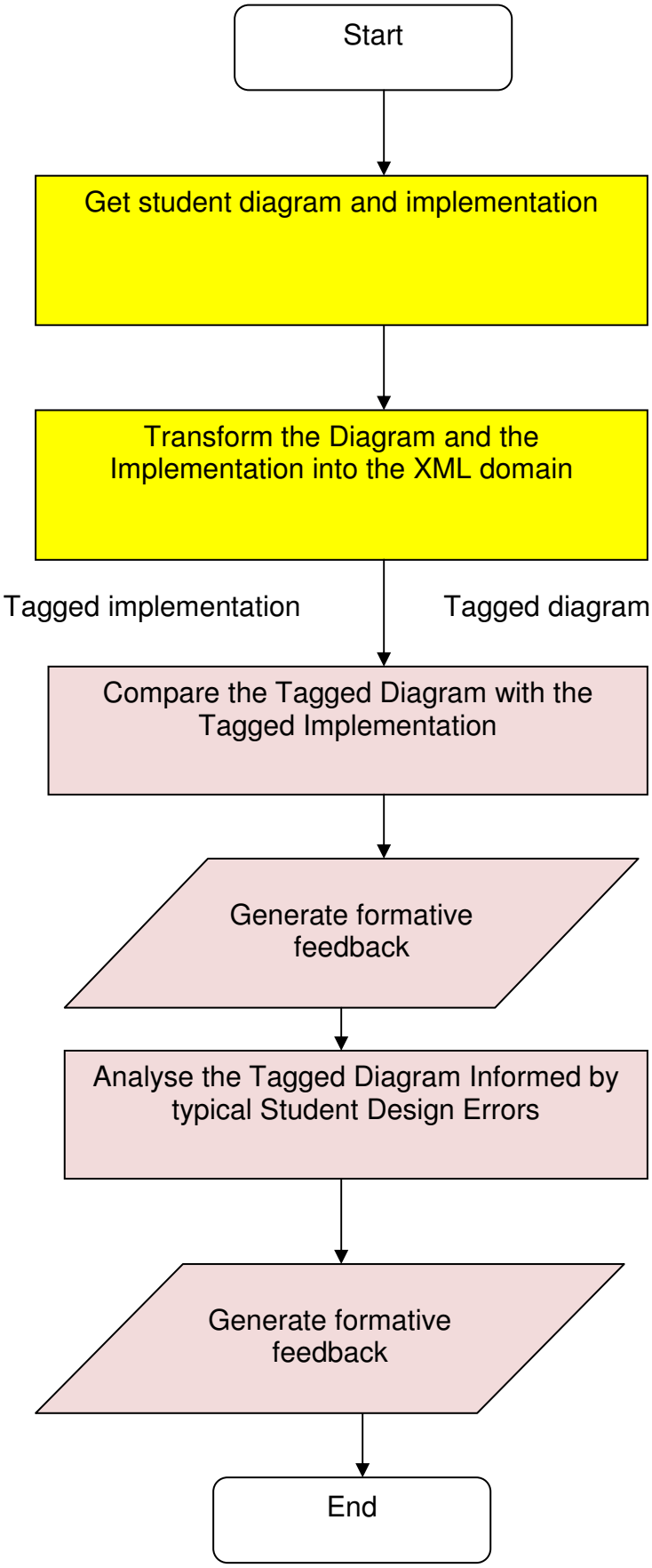


Figure 4.1 Overview Diagram of the Developed Assessment Tool

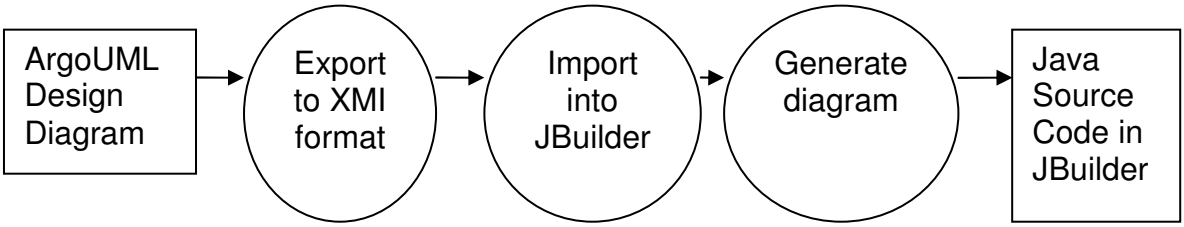
Automated

Manual

4.2 Inferred Artefacts through Forward and Reverse Engineering

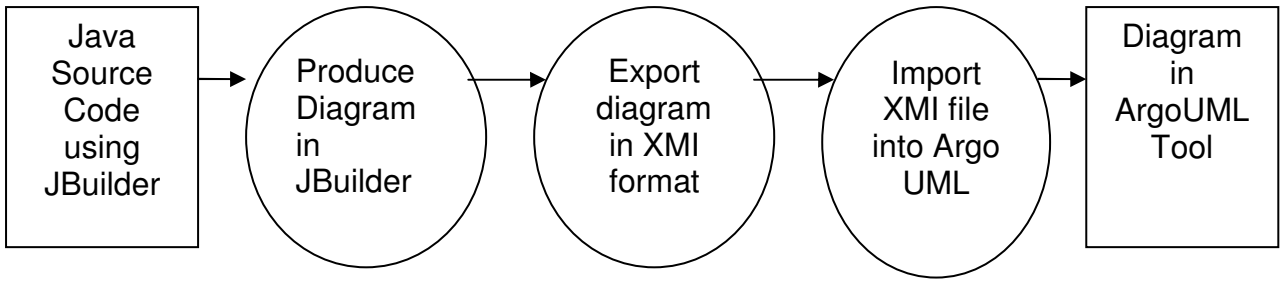
An experiment was undertaken to see the extent to which existing reverse and forward engineering tools could provide the basis for both producing inferred artefacts (as defined in chapter 3) and for describing an artefact's features. The resultant inferred artefacts were analysed to ensure that the originals' features were being preserved and that no erroneous or extraneous features had been added.

The assessment tool provides feedback upon the student submission. In our example, the students produced their UML diagrams using ArgoUML (Tigris, 2006), an open source CASE tool. They produced their Java source code using an IDE tool; either Eclipse (Eclipse Foundation, 2006) or JBuilder (Borland, 2008). An experiment was undertaken to investigate the round trip capabilities of these tools. The aim was to investigate their suitability for producing inferred artefacts and automating their description. A small Java program (using JBuilder) and a UML class diagram (using ArgoUML) was devised for testing purposes. This test program and its associated diagram contained features that would be typical of that expected from a student submission. The diagram and program comprised of a class inheritance hierarchy based around the concept of an employee and an additional class containing a main routine that interfaced with it. The main routine implemented a container relationship through the creation of a list of employees. Both tools supported the import and export of data through the XML Metadata Interchange (XMI) protocol (Object Management Group, 2007). The experiment consisted of forward engineering the UML diagram to produce source code (Figure 4.2) and reverse engineering the source code to produce a design diagram (Figure 4.3).



Inputs Design Diagram in Argo UML
Outputs Java Source Code
Intermediate output XMi representation of the design, Design diagram in JBuilder format

Figure 4.2 Forward Engineering: from Code to Diagram



Inputs Java Source Code
Outputs Design Diagram in ArgoUML
Intermediate output Design diagram in JBuilder format, XMi representation of the design,

Figure 4.3 Reverse Engineering: from Diagram to Code

The experiment raised some issues about the utilisation of these tools to produce inferred artefacts.

Forward engineering the design diagram successfully produced skeletal code for each class contained in the original diagram. The class names, attributes, methods and their parameters (with the exception of the constructors) were preserved in the process. It also preserved the inheritance hierarchy. However, the constructors of the child classes, the container relationship and its associated cardinality data were lost and not reflected in the inferred code.

Reverse Engineering the source code produced a diagram that preserved the number, name, methods, parameters and attributes of the classes. It also preserved the inheritance hierarchy. However, parameters into the constructors of the child classes and the container relationship were lost and not reflected in the diagram.

Hence, both reverse and forward engineering preserved the inheritance hierarchy and the signature of each class (though note the exception of parameters in class constructors). However, both processes failed to model the interaction (i.e. a container relationship). Further experimentation traced the problem to that of scoping for dynamically created objects. The XMI was not capturing the relationships embedded in the source code when the method of one class instantiated and created an object from another class.

An examination of the literature revealed that the problem of automatically reverse engineering a program's dynamic behaviour is a topic of ongoing research (Merdes and Dorsch 2006). This is particularly challenging for object oriented programs as the gulf between static specification and run-time behaviour is particularly wide (De Pauw *et al*. 1994). Features such as dynamic-binding and polymorphism pose limitations to static analysis (Lienhard *et al*. 2007). Hence, tools that analyse source code provide satisfactory results for static diagrams but their suitability for the dynamic behaviour of an application is limited (Merdes and Dorsch, 2006).

This was a disappointing find and problematic for developing the feedback tool. The static models produced by both reverse and forward engineering could have formed the basis upon which a semi-automated approach to generating feedback could have been developed. This would have required manually modifying the models to reflect the submission's dynamic behaviour. This semi-

automated approach was rejected as the resultant process would be too dependent upon the format and nuances of the models produced by the round-trip tools. Additionally, the pedagogic context was that of dynamically creating and manipulating objects. Hence, the tools used by the students could not be used to describe and infer artefacts from their submission. Consideration was given to finding or developing alternative tools. This was rejected as it would restrict the choice of tools that a tutor could ask the students to use. Consideration was also given to providing the students guidance on how to produce source code that circumvented this issue. However, the tool was meant to feed back to students on what they had done. It was considered to be pedagogically inappropriate to insist on a particular way of coding to ensure that the relationships were being picked up by the round trip process.

Hence, using round-trip engineering tools to infer and describe artefacts proved to be problematic. In retrospect, it would have been possible to remove the need for a round-trip tool and automate the description of the diagram and its source code as individual, separate entities. However, the focus of the research was upon comparison aspects of the tool and the evaluation of the feedback generated. Consequently, in order to progress the research, inferred artefacts were described by hand and produced through a manual analysis of the student submission.

4.3 Describing an Artefact's Features

Tagging is the mechanism by which an artefact's features are described in a format that the tool can recognise. It enables the tool to read and analyse the artefacts.

The eXtensible Markup Language (XML) was chosen as the language used to tag the artefacts because:-

- It is a standard developed and recognised by the world wide web consortium (http://www.w3.org).
- It enables the developer to describe and specify information with user-defined, meaningful labels.
- It supports the flexible, description and encapsulation of complex nested data structures. Thomas et al [2005] recognised that the imprecise nature of student diagrams necessitated a method for describing a diagram that was flexible and extendable.
- It provides an effective method for transferring data between systems.
- Open source Java routines are publicly available to developers to support the parsing of documents that contain data that has been described using XML.

A grammar was developed to tag the artefacts contained in the student submission. The grammar is bespoke and specific to our illustrative example. The application of the tool to other contexts and examples will require the development of an alternative grammar. However, the grammar is sufficiently generic to be applicable to most contexts that contain classes, objects and the relationships between them. In representing an artefact in a manner that facilitates such an extraction, tagging addresses the question of how to represent both a diagram and its implementation in a manner that enables the automation of a comparison to take place.

In this illustrative example the constituent features are:

- classes (including their names and their method and attribute signatures).
- relationships between these objects (including the type of relationship, associated direction and cardinality).

Artefacts from other contexts will contain different sets of features. However, the principle of adopting a tagging mechanism as a means of describing an artefact's features is one that can be applied to most multiple artefact contexts.

The tagging grammar developed is illustrated in Figure 4.4 below.

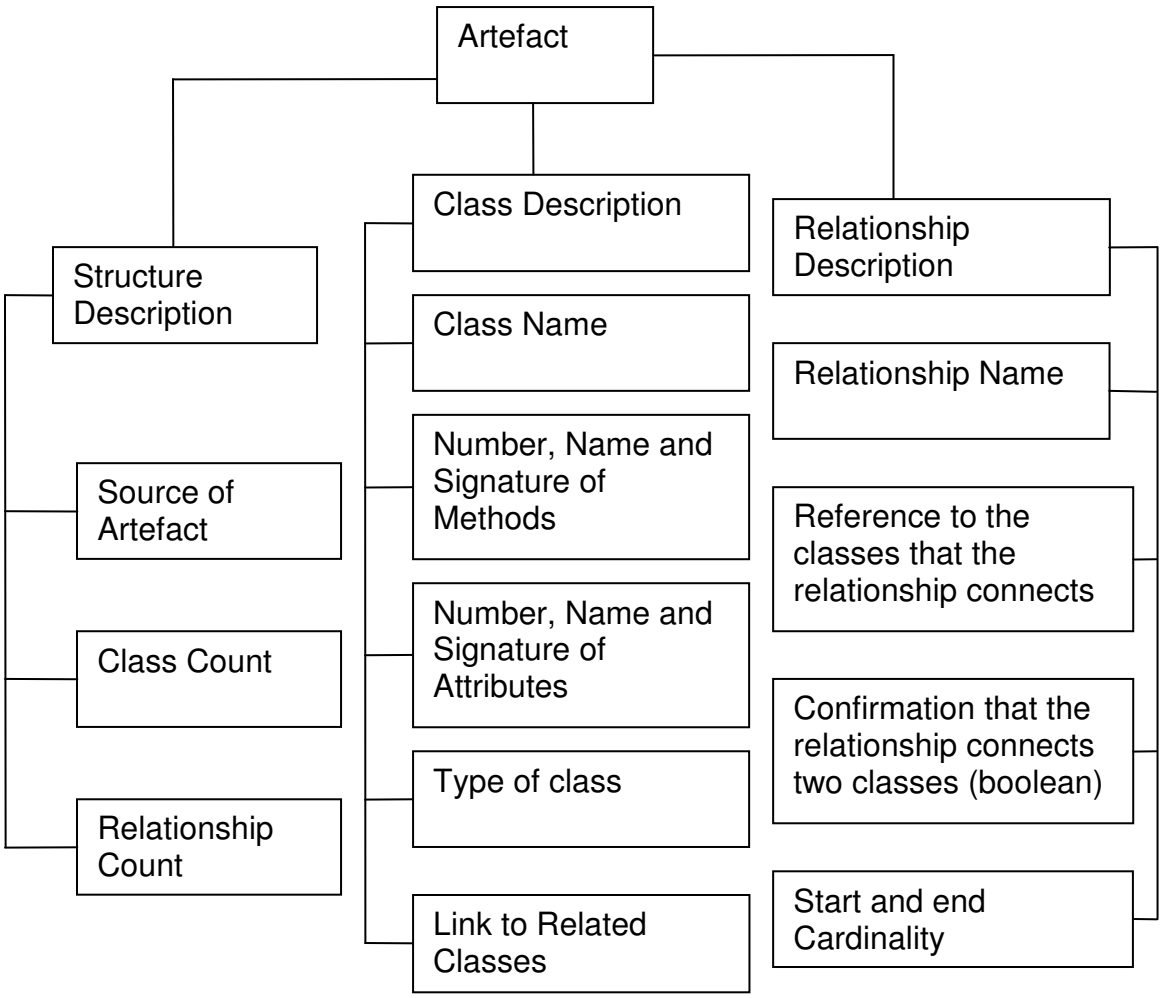


Figure 4.4 Diagram to Illustrate the Developed Tagging Grammar

The student submission had to be described in a manner that enabled

1. A comparison to be made between artefacts.
2. Typical errors made by students to be searched for and, if found, fed back upon.

Figure 4.5 illustrates those components of the tagging system that have been designed to specifically support the identification of typical design errors. It illustrates how the tagging grammar has been informed by the work of:

1. Thomasson *et al*. (2006) and Bolloju and Leung (2006) who identified a range of typical errors found in design diagrams produced by novice developers.
2. Tselonis *et al*. (2005) who identified a set of metrics that can describe diagrams in a manner that supports the matching of two diagrams

Feature	Source	Tag	Comment
Missing Cardinality Details	Bolloju and Leung (2006)	<relationship>	This tag identifies the relationship/association and its cardinality between the two classes
Incorrect Naming of Class	Bolloju and Leung (2006)	<class>	This tag contains a field that identifies the name of the class.
Incorrect Naming of Association	Bolloju and Leung (2006)	<relationship>	This tag identifies the type of relationship that is being used to link two classes.
Wrong Cardinality	Bolloju and Leung (2006)	<relationship>	This tag identifies the relationship/association and its cardinality between the two classes
Wrong Association	Bolloju and Leung (2006)	<relationship>	This tag identifies the relationship/association between two classes. It facilitates a check being made upon the two classes that have been linked and the type of relationship associated with the link.
Wrong location of attributes	Bolloju and Leung (2006)	<class> <attribute>	This tag contains fields that name the attributes of the class. Feedback upon this type of error can be generated through searching each class tag and comparing the name of the attributes with that being sought.
Wrong location of operations	Bolloju and Leung (2006)	<class> <method>	This tag contains fields that name the methods. Feedback upon this type of error can be generated through searching each class tag and comparing the name of the methods with that being sought.
Presence of derived or redundant attribute	Bolloju and Leung (2006)	<class> <attribute>	This tag contains fields that name each attribute and provides the total number of attributes.
Not all classes have been identified	Thomasson *et al*. (2006)	<structureDescription>	In their study 1 out of 180 students only managed to identify all 7 of the expected classes. Hence, this tag contains a "class count" field.

Figure 4.5 Table to illustrate how the tagging convention adopted supports typical student errors identified in the literature (continued overleaf)

Non referenced Class	Thomasson *et al*. (2006)	<class> <relationship>	59.9% of student submissions in their study exhibited this feature. A non-referenced class can be identified from the class tag (it is neither a parent nor a child) and the relationship tag (it does not appear at either end of a connector).
Reference to non-existing class	Thomasson *et al.* (2006)	<class> <relationship>	28.3% of the student submissions in their study exhibited this feature.
Single Attribute Misrepresentation	Thomasson *et al.* (2006)	<class> <attribute>	32.2% of the student submission in their study exhibited this feature. This is subsumed in Bolloju and Leung's [2006] 'wrong location of attribute'.
Multiple Attribute Misrepresentation	Thomasson *et al.* (2006)	<class>	Instead of identifying a separate class the student has identified the methods and attributes as components of another class. The class tag identifies the name and number of both attributes and methods.
Number of Incident Connectors (relationships) and their types	Tselonis *et al.* (2005)	<structureDescription> <relationship>	The number and types of relationships contained in the diagram can be determined from these tags
Component Type	Tselonis *et al.* (2005)	<class> <relationship> <structureDescription>	The number and type of components contained in the diagram can be determined from these tags.
Adjacent Components	Tselonis *et al.* (2005)	<class>	The number of components that each class is connected to is stored as a field within this tag.
Labels	Tselonis *et al.* (2005)	<class> <relationship>	Both these tags contain fields for a label/name.

Figure 4.5 Table to illustrate how the tagging convention adopted supports typical student errors identified in the literature

The tagging grammar divides an artefact into three main sections. These are:

1. A high level description of the artefact and its structure.
2. A description of the classes contained in the artefact.
3. A description of the relationships contained in the artefact.

The high level structure description consists of:

- The source of the artefact (design diagram or implementation).
- The number of classes contained in the artefact.
- The number of relationships contained in the artefact.

The description of each class contained in the artefact consists of:

- The name of the class.
- The number of methods and the name and signature of each method.
- The number of attributes and the name and signature of each attribute.
- The type of the class (parent, child, container, containee) and a reference to the respective related class(es).

The description of each relationship in the artefact consists of:

- The name of the relationship (inheritance, aggregation, dependency, association).
- A reference to the classes that the relationship connects.
- A confirmation that the relationship connects one class to another.
- A reference to the start and end cardinality of the relationship.

An example of a student design diagram, its tagged representation and a BNF grammar of the complete XML tagging grammar can be found in Figures 4.6, 4.7 and Appendix A respectively.

This section has presented the grammar developed for using XML to describe an artefact's features. It has highlighted how its development was informed by work in the literature on the identification of typical errors that students make in producing design diagrams. It noted that the grammar is bespoke to the

illustrative example. Two generic principles have been identified. The first was the need to transform the artefacts into a syntax understood by the tool. The second was the need to develop a grammar to enable the tool to analyse the features contained in the artefact.

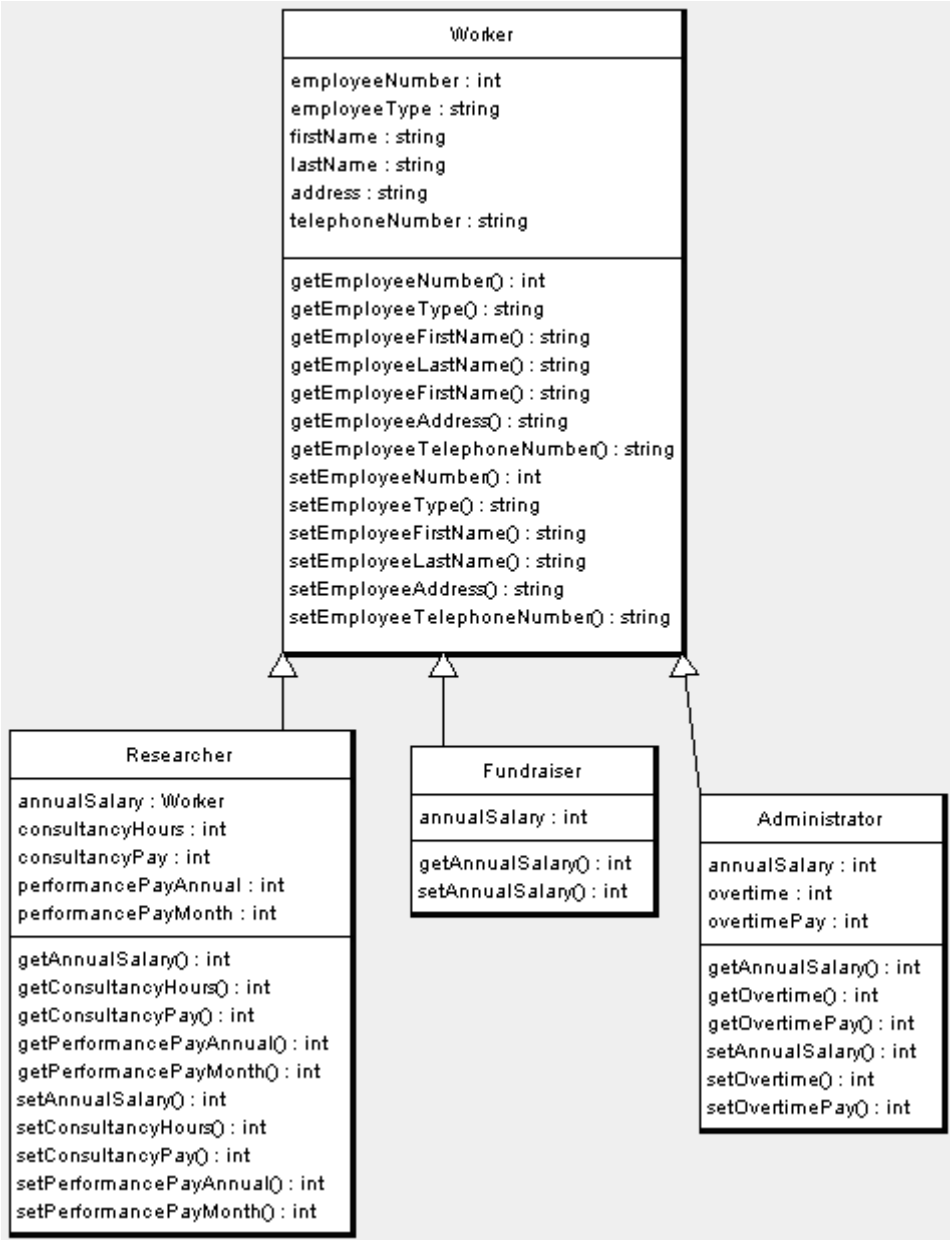


Figure 4.6 A Student-submitted UML Design Diagram

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<AML>
<comment>
Structure Diagram for sample New-07-08-02
Date Created 7th  July 2009
Structure Created from student-submitted diagram

</comment><StructureDescription source = "student diagram"  ClassCount = "4"
RelationshipCount = "3" />

<class id = "1" name = "Worker" attributeCount = "6" methodCount = "13" IsParent = "Yes"
childCount = "3" IsChild = "No" ParentCount = "0" IsContainer = "No" ContaineeCount = "0"
IsContainee = "No" ContainerCount ="0" AdjacentComponents = "3" AdjacentRef = "2 3 4" >

<attribute id = "att1.1" name = "employeeNumber"  type = "int" />
<attribute id = "att1.2" name = "employeeType"  type = "String" />
<attribute id = "att1.3" name = "firstName"  type = "String" />
<attribute id = "att1.4" name = "lastName"  type = "String" />
<attribute id = "att1.5" name = "address"  type = "String" />
<attribute id = "att1.6" name = "telephoneNumber"  type = "String" />


<method id = "meth1.1" name = "getEmployeeNumber" />
<method id = "meth1.2" name = "getEmployeeType" />
<method id = "meth1.3" name = "getEmployeeFirstName" />
<method id = "meth1.4" name = "getEmployeeLastName" />
<method id = "meth1.5" name = "getEmployeeFirstName" />
<method id = "meth1.6" name = "getEmployeeAddress" />
<method id = "meth1.7" name = "getEmployeeTelephoneNumber" />
<method id = "meth1.8" name = "setEmployeeNumber" />
<method id = "meth1.9" name = "setEmployeeType" />
<method id = "meth1.10" name = "setEmployeeFirstName" />
<method id = "meth1.11" name = "setEmployeeLastName" />
<method id = "meth1.12" name = "setEmployeeAddress" />
<method id = "meth1.13" name = "setEmployeeTelephoneNumber" />
<child id = "child1.1"  ClassId = "2" />
<child id = "child1.2"  ClassId = "3" />
<child id = "child1.3"  ClassId = "4" />
</class>

<class id = "2" name = "Researcher" attributeCount = "5" methodCount = "10" IsParent = "No"
childCount = "0" IsChild = "Yes" ParentCount = "1" IsContainer= "No" ContaineeCount ="0"
IsContainee = "No" ContainerCount ="0" AdjacentComponents = "1" AdjacentRef = "1" >

<attribute id = "att2.1" name = "annualSalary"  type = "UserDefinedType" />
<attribute id = "att2.2" name = "consultancyHours"  type = "int" />
<attribute id = "att2.3" name = "consultancyPay"  type = "int" />
<attribute id = "att2.4" name = "performancePayAnnual"  type = "int" />
<attribute id = "att2.5" name = "performancePayMonth"  type = "int" />

<method id = "meth2.1" name = "getAnnualSalary" />
<method id = "meth2.2" name = "getConsultancyHours" />
<method id = "meth2.3" name = "getConsultancyPay" />
<method id = "meth2.4" name = "getPerformancePayAnnual" />
<method id = "meth2.5" name = "getPerformancePayMonth" />
<method id = "meth2.6" name = "setAnnualSalary" />
<method id = "meth2.7" name = "setConsultancyHours" />
<method id = "meth2.8" name = "setConsultancyPay" />
<method id = "meth2.9" name = "setPerformancePayAnnual" />
<method id = "meth2.10" name = "setPerformancePayMonth" />

<parent id = "parent2.1"  ClassId = "1" />
</class>
```

```
<class id= "3" name = "Fundraiser" attributeCount = "1" methodCount = "2" IsParent = "No"
childCount= "0" IsChild = "Yes" ParentCount = "1" IsContainer = "No" ContaineeCount = "0"
IsContainee = "No" ContainerCount = "0" AdjacentComponents = "1" AdjacentRef = "1" >

<attribute id = "att3.1" name = "annualSalary"  type = "int" />

<method id = "meth3.1" name = "getAnnualSalary" />
<method id = "meth3.2" name = "setAnnualSalary" />

<parent id = "parent3.1"  ClassId = "1" />

</class>

<class id= "4" name = "Administrator" attributeCount = "3" methodCount = "6" IsParent = "No"
childCount= "0" IsChild = "Yes" ParentCount = "1" IsContainer = "No" ContaineeCount = "0"
IsContainee = "No" ContainerCount = "0" AdjacentComponents = "1" AdjacentRef = "1"  >

<attribute id = "att4.1" name = "annualSalary"  type = "int" />
<attribute id = "att4.2" name = "overtime"  type = "int" />
<attribute id = "att4.3" name = "overtimePay"  type = "int" />

<method id = "meth4.1" name = "getAnnualSalary" />
<method id = "meth4.2" name = "getOvertime" />
<method id = "meth4.3" name = "getOvertimePay" />
<method id = "meth4.4" name = "setAnnualSalary" />
<method id = "meth4.5" name = "setOvertime" />
<method id = "meth4.6" name = "setOvertimePay" />

<parent id = "parent4.1"  ClassId = "1" />
</class>

<relationship id = "rel1" name = "inheritance" nondangling = "BothEndsConnected" startclassid =
"1" startcardinaility = "none" endclassid = "2" endcardinality = "none" />

<relationship id = "rel2" name = "inheritance" nondangling = "BothEndsConnected" startclassid=
"1" startcardinality = "none" endclassid = "3" endcardinality = "none" />

<relationship id = "rel3" name = "inheritance" nondangling = "BothEndsConnected" startclassid=
"1" startcardinality = "none" endclassid = "4" endcardinality = "none" />

</AML>
```

Figure 4.7 The Resultant Tagged Student Diagram

4.4 A Heuristic for Comparing Artefacts and Feedback Generation

This section describes the heuristic developed to compare two artefacts and generate formative feedback. The underpinning pedagogic aim is to feed back to the student upon their submitted design and implementation. The rationale for doing so is that the student can reflect upon their adherence to the software development lifecycle and their understanding of the relationship between design and implementation abstractions. The feedback consists of positive reinforcement in addition to identifying where mistakes have been made and further learning is

needed. Feedback is generated at two levels: holistic and specific. Holistic feedback reports the total number of features in the artefacts and the number for which a match could/could not be found. Specific feedback contains details on their comparison.

The approach adopted is to initially compare the two submitted artefacts and identify their consistent and superfluous features (terms defined in chapter 3). Consistent features are positively reinforced. Guidance for further learning is provided for the superfluous features. This approach poses several challenges including:

- How do you compare the artefacts and identify the consistent and superfluous features?
- What criteria do you use to determine consistent and superfluous features?
- How do you produce feedback that is pertinent to the student's context from a generic heuristic that compares features contained in artefacts?

The heuristic consists of visiting each feature of one artefact and comparing it with all features of the other. The output of the comparison is a matching score and a list of feedback comments for each pair of features. The matching score is a metric used to indicate the extent to which the features match. A high score indicates a strong match; a low score indicates little similarity. The list of feedback comments are generated during the calculation of the matching score and provide detail of why it is that a feature pair has produced a high/low matching score. Feedback for high scores takes the form of positive reinforcement and for low scores directs the student towards further learning. Feedback for scores that are neither high nor low contain a balance of developmental and reinforcing comments. This balance is determined by the calculated score.

The matching score is a calculated metric describing how well the features of the two artefacts compare. The score ranges from 0 (representing no match) to 10 (full match). Its calculation is based on a comparison between the artefacts' features. In this example, these features are the classes and the relationships between them. The matching score for two classes is determined by comparing their signatures. The signature consists of the class name, its attributes and its methods. It is calculated by the following formula:

MatchingScore = (ScoreOnClassNames + (ScoreOnMethods + ScoreOnAttributes)/2) /2

where

- the ScoreOnClassNames metric is a value that ranges from 0 to 10. If the two names match exactly the metric is 10.
- The ScoreOnAttributes and ScoreOnMethods metrics are values that range from 0 to 10. Each is calculated as a function of the name and number being the same for both classes.

A higher weighting is allocated to the class names as it is a particularly strong identifier given that the student has named the class in both the diagram and implementation. This would not necessarily be the case when a comparison is being made with a class name coming from a tutor-supplied solution and one that came from the student submission. At the point of calculating the score, feedback comments are generated and stored in a table. Positive reinforcement is stored for high scoring constituent parts whilst lower scores store developmental feedback. For example, a comparison might result in a high score on the class methods but a low score on attributes. Feedback would be that the interface between classes is understood (reinforcement) but that more work needs to be undertaken in modelling an object's data (developmental). Figure 4.8 and 4.9 below illustrates how the matching score is determined for comparing the methods contained in each class. The tool currently has the

tolerance values, feedback comments and resultant matching scores embedded within it. Tutor modification of these parameters would be possible through a tool that enabled change of the parameters detailed in Appendix B.

Test Val	Feedback	Matching Score
The number of methods in class 1 is equal to the number of methods in class 2 AND The number of the methods in class 1 >0 and the number of methods in class 2 >0	These two classes have the same number of methods.	methodCountScore =10
The number of methods in class 1 is equal to the number of methods in class 2 +- methodCountTolerance AND The number of the methods in class 1 >0 and the number of methods in class 2 >0	These two classes differ slightly in the number of methods that each contains.	methodCountScore = 5
The number of methods in class1 is different to the number of methods in class 2 (outside the methodCountTolerance) AND The number of the methods in class 1 >0 and the number of methods in class 2 >0	There is a significant difference in the number of methods specified for each class.	methodCountScore = 0
The number of methods in class1 is zero OR The number of methods in class2 is zero	One of your classes does not contain any methods. This suggests that you probably need to revisit your notes on how you identify the methods of a class	methodCountScore = 0
The number of methods in class1 is zero OR The number of methods in class2 is zero	Neither of these two classes contain any methods. This suggests that you probably need to revisit your notes on how you identify the methods of a class.	methodCountScore = 0

Figure 4.8. Table to illustrate how the matching score for the number of methods contained in a class is determined

Test Val	Feedback	Matching Score
The number of methods in class 1 is equal to the number of methods in class 2 AND The names of the methods in class 1 are equal to the names of the methods in class2	There is a good match for both the method name and number for these two classes.	methodNameScore =10
The number of methods in class 1 is equal to the number of methods in class 2 +- methodCountTolerance AND The names of the methods in class 1 are equal to the names of the methods in class2 +- methodCountTolerance	These two classes match well in their methods both on name and number with only minor differences between the two.	methodNameScore = 7
The number of methods in class1 is different to the number of methods in class 2 (outside the methodCountTolerance) AND All the names of the methods in class 1 are equal to the names of a subset of the methods in class 2 (where the number of methods in class 1 is less than the number in class2)	Some of the methods match well in these two classes but a significant number don't. You probably need to visit your notes on analysis and design and look again at how you identify the methods of a class.	methodNameScore = 5
The number of methods that match in name and number are less than the tolerance	The methods described in these two classes suggest that you think these are very different entities. You need to revisit your notes on identifying and implementing objects.	methodNameScore = 0

Figure 4.9. Table to illustrate how the matching score for the names of the class methods is determined

The **methodCountTolerance** is set to a value of 2. This value was chosen as too high a value could result in a (false-positive) high matching score.

The formula for calculating the overall score on methods is:

ScoreOnMethods = (methodCountScore + methodNameScore)/2

The formula for calculating the overall score on attributes is similarly calculated and is detailed in the user handbook (Appendix B).

Figure 4.10 illustrates the data structure used by the tool to store the calculated matching scores and feedback comments. It is referred to as a feedback table. In this example both artefacts contain a (differing) number of class features.

Artefact1	**Artefact 2**				
	Class 1	**Class 2**			**Class T**
Class 1	MatchData11	MatchData12			MatchData1T
Class 2	MatchData21	MatchData22			MatchData2T
Class S	MatchDataS1	MatchDataS2			MatchDataST

Class S
Class 1
Score On Class Names
Score On Attributes
Score On Methods
Overall Matching Score
The Classes Match with each other (boolean)
List of Feedback Comments

Figure 4.10 Diagram to Illustrate the Feedback Table when comparing two artefacts

The rows of the table are indexed by the number of classes in the first artefact and the columns by the number in the second. Each element of the table stores the data for the classes being compared, a matching score and a list of feedback comments. The list is populated during the comparison.

Minimal stemming was used when comparing names. This reason for this was that, in our example, both artefacts are being produced by the same student. The student has decided what to call the features contained in both the code and

diagram artefacts. If our example involved comparing a student diagram with one produced by a tutor then a more sophisticated stemming, or similar, technique such as that advocated by Thomas *et al*. (2009) or Jayal and Shepperd (2009) would need to be deployed.

Once all class pairs have been compared they are revisited. A class from one artefact could have matched well with several different classes from the other. The next step is to identify the best match for each pair. This is done by identifying the pair with the highest matching score. A threshold value is set for the score (currently set at 7 from a maximum of 10). Two classes are considered to have matched only if their score exceeds this value. Feedback is generated by iterating through the list of comments for the class pair contained in the feedback table. This feedback could be developmental, reinforcing or a combination of both as it will have been determined at the point at which the comparison was made. Classes from either artefact which fall below the threshold are those for which a match could not be found. These are reported and developmental comments are fed back to the student. A similar approach is taken to compare the relationships contained in the artefacts.

4.5 Searching for Typical Errors

In addition to comparing artefacts the developed tool supports the generation of feedback through an analysis of a single artefact. In our example, analysis of the student diagram in isolation was restricted to searching for a subset of typical errors made by students when developing design diagrams. Specifically, two common errors (Thomasson *et al.* (2006) and Thomas *et al*. (2007)) made at the design stage were searched for:

- Classes in the diagram are not related to any other components (isolated/extraneous).

- Relationships contained in the diagram do not connect two classes (dangling).

An expanded list of errors could have been produced from the literature or alternatively produced locally by a tutor. In either case the heuristic would need enhancing. However, for the context of this illustrative example the list was restricted to that indicated above. A flow chart describing the heuristic developed is presented in Figure 4.11.

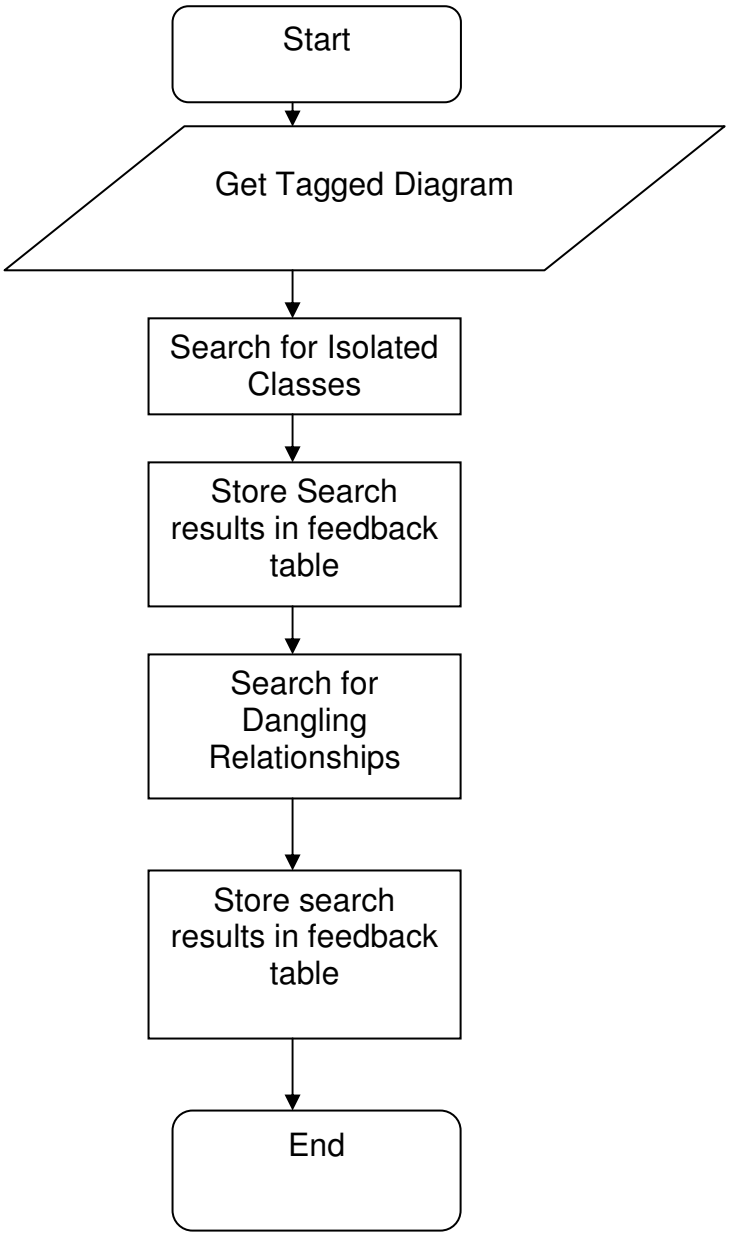


Figure 4.11 Flow chart of the Heuristic to Analyse a Diagram in Isolation

The feedback generated is illustrated in Figure 4.12. It shows the list of errors/conditions the heuristic searched for and the feedback, supplied by the tutor, if the errors/conditions were found.

Condition	**Feedback**
Diagram Contains Isolated Classes	At least one class in your design diagram is shown not to be related to any others. You need to do some further reading on how a program that consists of message passing objects works.
Diagram Does Not Contain Isolated Classes	Your diagram does not contain any isolated classes. Well done. This shows that you understand that a program works through objects being related to each other.
All relationships in the Diagram are appropriately connected	All of the relationships that you have identified have a start class and an end class. This is good as it shows that you have understood that relationships are used to connect the classes contained in your diagram.
Diagram Contains one or more Dangling Relationships	You have drawn a relationship that does not connect two classes. You need to revisit how you identify and represent relationships between objects.
Diagram Does not Contain Any Relationships	Your design diagram does not contain any relationships. You need to revisit your understanding of object orientation and how objects are related to each other.

Figure 4.12 Table to show the feedback generated by the tool when analysing the student diagram

Whilst the feedback generated is context specific to our illustrative example the general principle here is that a mechanism is required to:

1) identify the conditions upon which feedback needs to be generated.

2) specify the feedback to be given to the student when these conditions are met.

4.6 An example

An illustrative example is presented below based on a student design diagram (Figure 4.13), a diagram inferred from the student code (Figure 4.14) and the feedback generated by the heuristic (Figure 4.15). The tool generates nine feedback comments. Comments 1 and 2 are produced by searching for tutor specified errors as discussed in section 4.5, comments 3 to 5 constitute holistic feedback and 6 – 9 specific feedback as discussed in section 4.4.

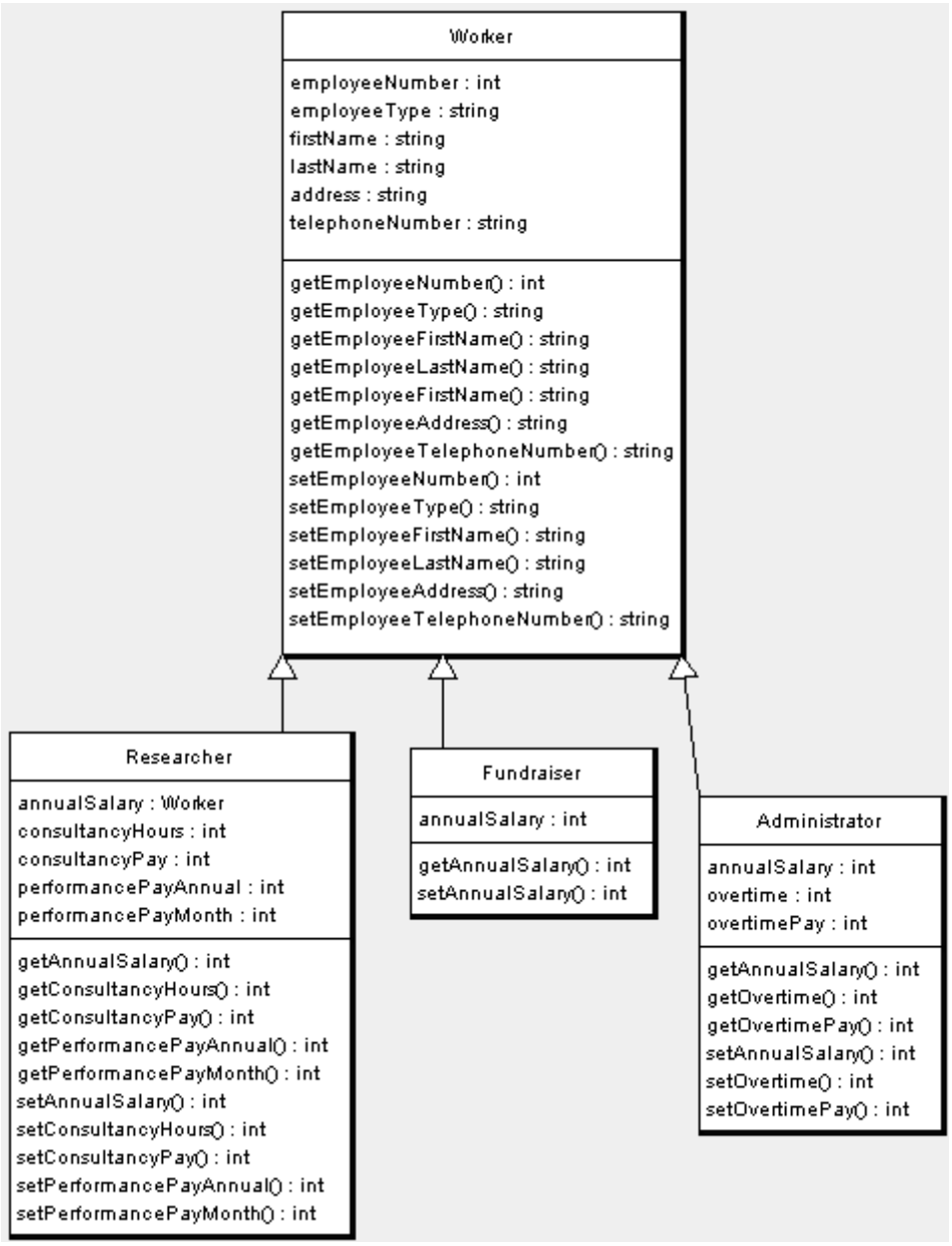


Figure 4.13 A Submitted Student Design Diagram

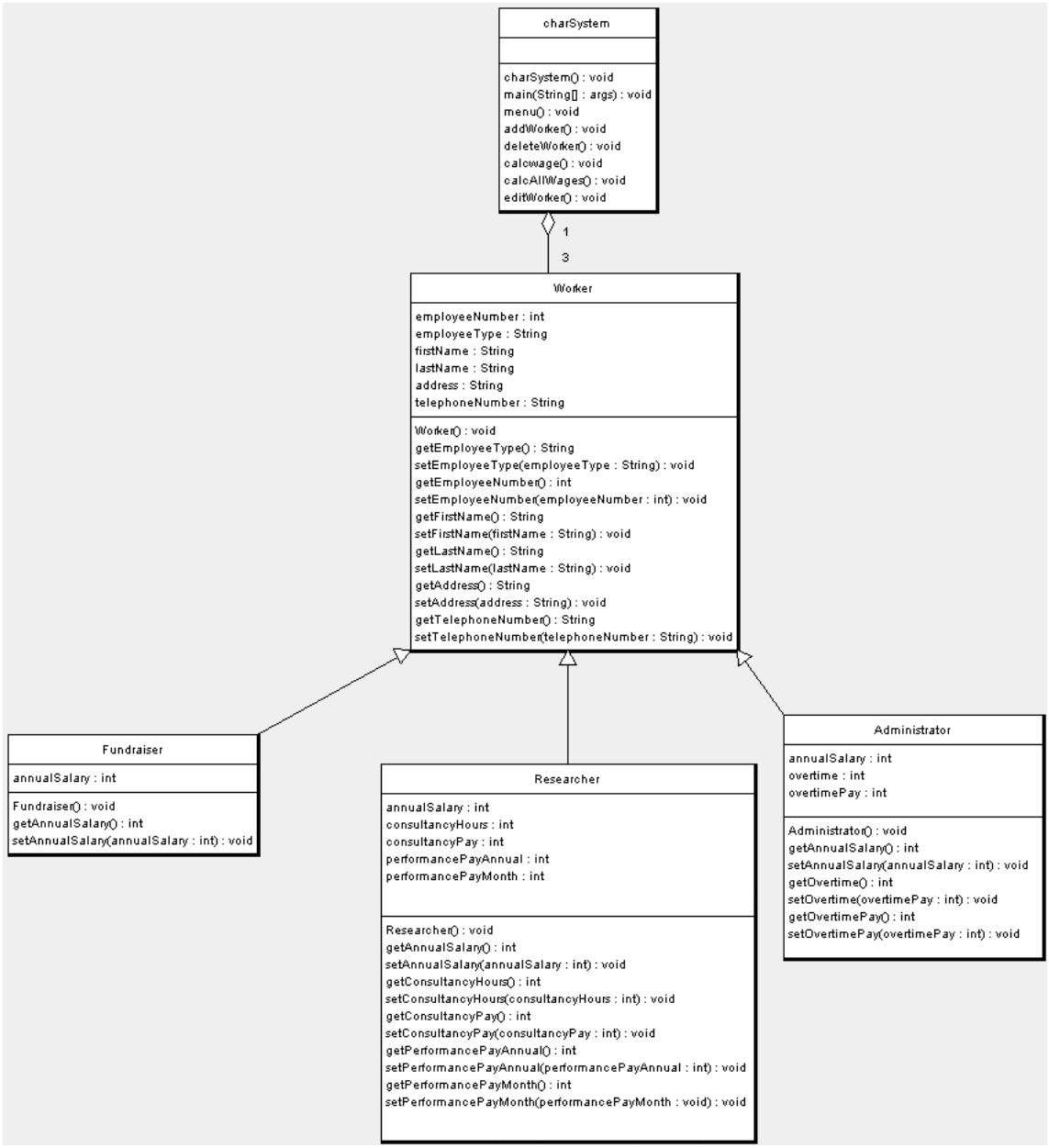


Figure 4.14 The Diagram inferred from submitted source code.

Feedback Ref	Feedback 28
Comment 1	Your diagram does not contain any isolated classes. Well done. This shows that you understand that a program works through objects being related to each other
Comment 2	All of the relationships that you have identified have a start class and an end class. This is good as it shows that you have understood that relationships are used to connect the classes contained in your diagram.
Comment 3	The structure contained in your diagram is topologically close to that contained in your implementation. This means that there is a good match between your design diagram and your implementation.
Comment 4	The number of Classes in your Design Diagram is 4 and in your implementation you have 5 (9 in total) There are 4 classes that match well when comparing your design with your implementation (8 from 9) There is 1 class for which a match could not be found
Comment 5	The number of Relationships in your Design Diagram is 3 and in your implementation you have 4 (7 in total) There are 3 relationships that match well when comparing your design diagram with your implementation (6 from 7) There is 1 relationship (from 7) for which a match could not be found
Comment 6	Class Worker from your program is a close match to Class Worker from your design The names of these two classes match well Both classes contain the same number of attributes The attributes in these two classes match well on both name and number These two classes have the same number of methods Class Fundraiser from your program is a close match to Class Fundraiser from your design The names of these two classes match well Both classes contain the same number of attributes The attributes in these two classes match well on both name and number These two classes differ slightly in the number of methods that each contains

	Class Researcher from your program is a close match to Class Researcher from your design The names of these two classes match well Both classes contain the same number of attributes The attributes in these two classes match well on both name and number These two classes differ slightly in the number of methods that each contains Class Administrator from your program is a close match to Class Administrator from your design The names of these two classes match well Both classes contain the same number of attributes The attributes in these two classes match well on both name and number These two classes differ slightly in the number of methods that each contains
Comment 7	Your implementation contains a class called charSystem which is sufficiently different from all those contained in your design diagram to suggest that there is a mis-match between what you have designed and what you have implemented
Comment 8	You have shown that you understand how to implement the relationships that you have identified in your design. Well done You have shown this through :- Your design and program both relating class Worker and class Fundraiser with a inheritance relationship Your design and program both relating class Worker and class Researcher with a inheritance relationship Your design and program both relating class Worker and class Administrator with a inheritance relationship
Comment 9	The aggregation relationship in your program that connects class charSystem with class Worker Could not be matched with any relationship in your design. You need to think about how your design matches your implementation for all classes and objects contained in your system

Figure 4.15 The Feedback generated by the tool following an analysis of the submitted student design diagram (Figure 4.12) and source code (Figure 4.14).

Figure 4.16 below presents a pseudo-code description of how the tool generates the feedback illustrated in Figure 4.15.

```
1. Upload_artefacts;
2. Decode_artefacts_into_internal_java_based_data_structures;
3. search_student_diagram_artefact_for_typical_novice_errors;
4. generate_feedback_on_structure;
5. For each feature in artefact 1
6.     For each feature in artefact 2
7.         compare_features;
8.         compute_matching_ score_and_access_score_related_feedback_comment;
9.         store_matching_score_and_feedback_for _the_artefact_pair;
10.    End inner_loop;
11. End outer_loop;


12. For each feature in artefact 1
13.   Identify_the_feature_in_artefact2_with_the_highest_matching_score;
14.   If (highest_maching_score >= threshold)
15.      set_matching_boolean_flag_to_true_for_ this_feature_pair;
16.      increment_count_for_number_of_matching_features;
17.    else
18.       set_matching_boolean_flag_to_false_for_this_feature_pair;
19.    end if;
20. end loop;
21. output_holistic_feedback;
22. output_detailed_feedback_on_matching_features;
23. output_detailed_feedback_on_nonmatching_features;
```

Figure 4.16 A pseudo-code description of how the tool generates feedback

A user manual for the tool has been provided in Appendix B. This provides further detail upon the feedback comments, the test conditions under which they are generated and the details of how the matching scores for the artefact's features have been calculated. Below is a line-by-line description of the pseudo-code illustrated in Figure 4.16.

Line 1

The tool requires as input two artefacts described using the grammar defined in Figure 4.4.

Line 2

This routine utilises imported Java routines (McLaughlin 2001) to use the Document Object Model (www.w3.org/DOM/) to extract the features described in the two artefacts. The result is to populate two internal tool-specific lists: a list of features contained in artefact 1 and a separate list for artefact 2. In our worked example this relates to the student design diagram and its implementation respectively.

Line 3

This routine accesses the features contained in the student diagram and searches for common diagrammatic errors as described in Figures 4.11 and 4.12 respectively. It generates feedback based upon the presence/absence of these errors. In Figure 4.15 of our worked example, this relates to feedback comments 1 and 2 respectively.

Line 4

This routine provides feedback as a result of comparing the structure of the two artefacts . In Figure 4.15 of our worked example, this relates to feedback comment 3. The table below indicates the test condition and the feedback generated.

Test Val	Feedback Comment
classCount1 == classCount2 && relCount1 == relCount2	The structure contained in your diagram is topologically equivalent to that contained in your code. This means that there is a strong match between your design diagram and your implementation.
diff(classCount1, classCount2) <= classCountTolerance && diff(relCount1, relCount2)<=relCountTolerance	The structure contained in your diagram is topologically close to that contained in your code. This means that there is a good match between your design diagram and your implementation.
diff(classCount1, classCount2) + diff(relCount1, relCount2) > (classCountTolerance + relCountTolerance)	There are significant differences in the structure of your design diagram when compared to your code. You need to do some more reading on the software development lifecycle and the relationship between design and implementation.

Key

classCount1 = number of class features contained in artefact 1.

classCount2 = number of class features contained in artefact 2.

relCount1 = number of relationship features contained in artefact 1.

relCount2 = number of relationship features contained in artefact 2.

classCountTolerance = 1

relCountTolerance = 1

Lines 7 to 9

These routines compare the signature of two features, one described in artefact 1 and the other in artefact 2. A matching score is calculated as described in section 4.4. Figure 4.8 and 4.9 illustrates the feedback generated for each score. These feedback comments are pre-determined and uploaded at run time. Consequently should a tutor wish to change comments or apply the tool to a different context he/she would need to provide an alternative set of comments for the features and their matching scores.

The routine on line 9 stores the relevant scores and feedback comments in a feedback table as detailed in Figure 4.10.

These routines generate feature-specific feedback comments. In Figure 4.15 of our worked example, these are feedback comments 6 to 9.

Line 12-20

These routines identify the highest matching score for the features contained in each artefact pair. If this score is greater than or equal to a threshold value (currently set to a value of 7 from a maximum score of 10) then the artefact pair are considered a match and the boolean match_found flag for this feature pair is set to true. If the highest matching score for the artefact pair is less than the threshold, the flag is set to false.

Line 21

This routine produces the holistic comments 4 and 5 in Figure 4.15. The routine uses the number of matching artefacts found (Line 16) and the flag set for each matching feature pair (Line 18) to provide feedback on the total number of features contained in each artefact and how many matching features were detected.

Line 22

This routine accesses the matching_boolean_flag for each artefact pair (the value of this flag is set on line 9). Where the flag is true, the routine accesses the stored feedback strings (these were determined and set by the routine on line 9) and outputs them to the student's feedback file. In Figure 4.15 of our worked example, these are feedback comments 6 and 8.

Line 23

This routine identifies those features from both artefacts for which a match has not been found. The routine names the features and outputs a predefined feedback comment to the student's feedback file. In Figure 4.15 of our worked example, these are feedback comments 7 and 9.

4.7 Summary

This chapter has presented the development of an automated feedback tool applicable to a two-artefact submission (a diagram and some Java source code). The tool was illustrated by applying it to a specific example.

A heuristic was presented that identified the similarity between features contained in two artefacts. Generating formative feedback based upon similarity is challenging as there are aspects of the student submission that match well and those which are erroneous. What is needed is the generation of positive reinforcement for those features that match well and developmental feedback for those that do not. The solution presented measures the similarity between features by calculating a matching score. The higher the score the greater the similarity between the features. The method of linking different feedback comments to a specific (range) of matching scores provides the means for discriminating between developmental and reinforcing feedback. It offers several benefits including:

- It enables a blend of reinforcing and developmental feedback to be generated for the student submission.
- In principle it enables a distinction to be made between the generation of context-specific feedback and a generic heuristic that compares features contained in artefacts.

The chapter identified several generic principles that emerged from the example presented:

- There is a need to transform all the artefacts into a single format (a representation having a defined syntax) recognised by the tool.
- A tagging mechanism is required to perform the transformation.
- A grammar is required to describe an artefact's features.

Capturing the dynamic behaviour embedded within the submission proved to be more challenging than originally anticipated. The decision to manually tag the submission was taken to expedite this research. How to capture dynamic behaviour embedded in the student submission in a format that enables the automation of formative feedback is an item of further work discussed in chapter 7.

The next chapter discusses the evaluation of the feedback tool. This involved applying the tool to a sample of student submissions. Evaluation of the feedback generated was undertaken by both a group of students and members of the computer science education community.

Chapter 5. Evaluation Methodology

5.1 Introduction

The previous chapters have discussed how automating the assessment of diagrams could be extended to the case where a student submits a design diagram with an accompanying implementation. They describe the development of a proof-of-concept tool that takes the student submission as input and automatically generates formative feedback. This chapter details how the effectiveness of this approach has been evaluated. Evaluation has focused upon the formative assessment feedback provided by this approach and this chapter discusses the methodology adopted for the evaluation. Diagrams, with their implementations, were collated and divided into two sets: one reserved for experimentation and development of the tool and the other reserved exclusively for evaluation. Both the student voice and a set of human markers, taken from members of the computer science education community, were included in the evaluation process via questionnaires. The chapter examines the issue of variability between individual markers and discusses the steps taken in the design of the evaluation to mediate against this.

Section 5.2 provides an overview of the experimental approach taken. Section 5.3 discusses the data collated and how it was divided into experimental and evaluative sets. Section 5.4 discusses how a set of both summative grades and formative comments were generated from the evaluative data set by a team of human markers. Section 5.5 discusses how variations in the marking by individual markers were considered. Section 5.6 discusses the design and development of the two questionnaires that were used in the evaluation of the formative comments generated by the tool. It describes how the first questionnaire was used to

undertake a comparison between human-generated comments and those generated by the tool and how the second was used to solicit student input into the evaluation. Section 5.7 discusses how variations in the evaluation of feedback comments by individual evaluators were considered whilst section 5.8 discusses how the first questionnaire returns were analysed to evaluate the feedback comments generated by the tool. Section 5.9 discusses how the second questionnaire was used to gain the students' evaluation of the tool-generated feedback.

5.2 Overview of the Evaluation Process

This section provides an overview of the evaluation process adopted in the evaluation of this research with later sections providing the detail.

The focus of the experimentation is to evaluate the effectiveness of the automatically generated formative feedback comments. This is complicated because humans often do not agree on what constitutes good marking or what constitutes good feedback (Yorke, 2003). Therefore, it was decided to compare automatically-generated feedback against human-generated feedback. If the automatically-generated feedback was at least as good as the human-generated feedback, it can be said that the tool generates appropriate and adequate feedback.

There were two main phases in the evaluation. Phase 1 collected feedback comments from both the tool and a team of expert markers. Phase 2 evaluated the tool's comments. In particular:

Phase 1 consisted of:

1. The collection of a corpus of suitable student submissions.
2. The collection of feedback on the submissions generated by both the automatic process and several human expert markers.

and phase 2 consisted of:

1. The evaluation of the feedback by human domain experts to determine its quality.
2. The evaluation of the feedback by students based upon the work that they submitted.

Figure 5.1 provides a summary of all the steps in the approach. The approach is thorough but complex. It is scalable in both number and complexity of the student submission. However, it is reliant on the evaluators having the time both to mark the student submission and to evaluate the feedback comments. Identifying, and soliciting the co-operation of, the human expert markers needed to make this time commitment manageable will be a challenge if the scale were to be increased. Experiments of this nature found in the literature are often based on small student samples (one or possibly two cohorts) and a small number of markers. This experiment is large by comparison. The timescales were manageable for the majority of the evaluators.

Evaluating the feedback required a comparison between tool and human generated comments and consequently three experiments to take place:-

- An experiment to test for significant differences between summative grades generated by a team of markers.

- An experiment to test for significant differences between members of a team of evaluators who had rated formative feedback comments.
- An experiment to test for significant differences in the evaluative ratings for the tool-generated comments when compared to those that were human-generated.

The sections below discuss these experiments and Appendix E presents the detail of the statistical methods deployed.

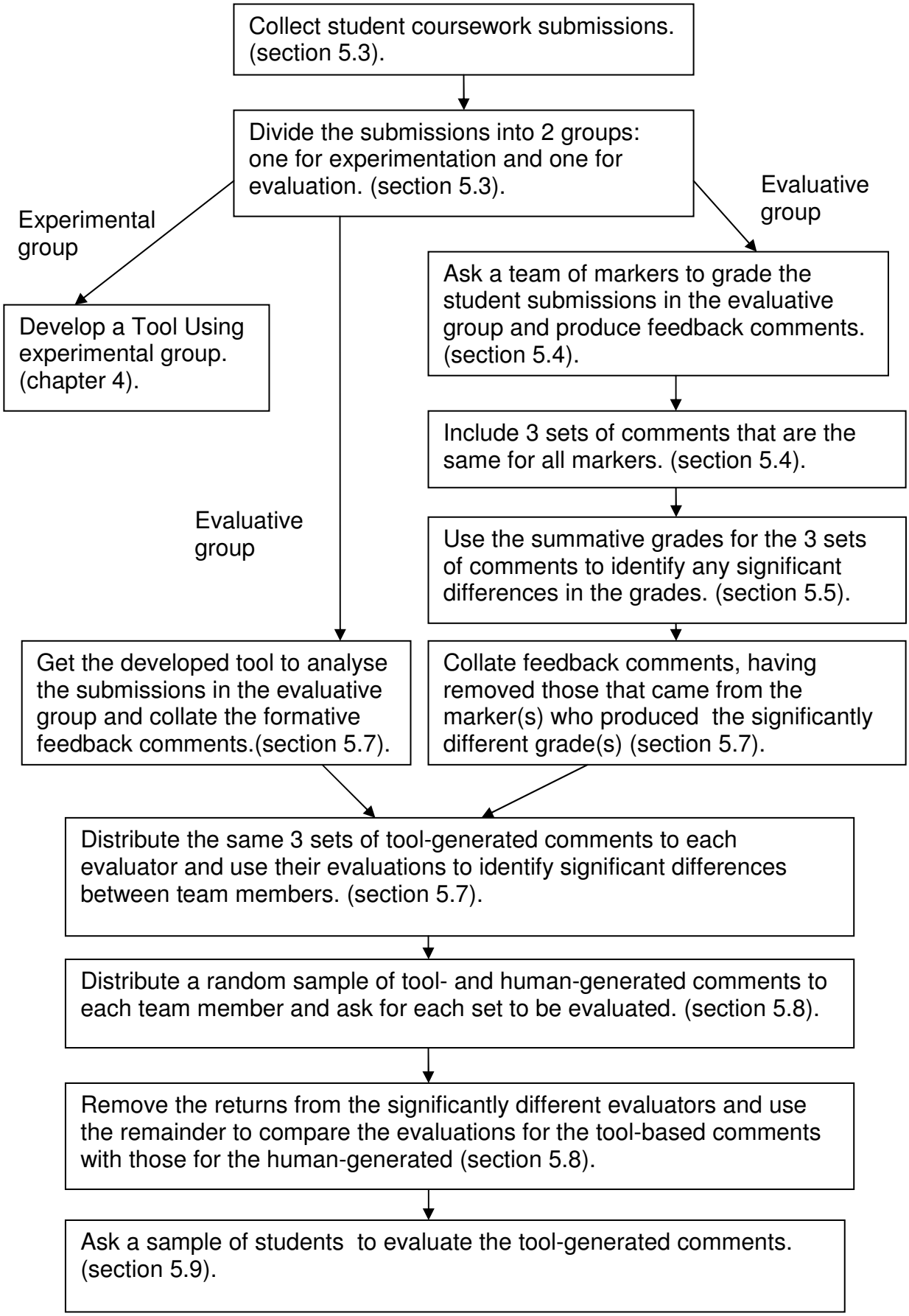


Figure 5.1 Diagram to illustrate the process of comparing tool-generated comments with those that were human-generated.

5.3 Student Submission Data

This section presents an overview of the data that has been collated in order to evaluate the tool. The division of the data into two sets, one for experimentation and one for evaluation is discussed.

Six data sets consisting of student submissions were collated over four years and two Higher Education Institutions. Each student submission consisted of a design diagram and its accompanying implementation. Appendix C (part 3) provides an example assignment brief. All briefs used in this research were authentic and real assignments i.e. they have been used in the assessment of the undergraduates' understanding of object orientation. Consequently, they have been subject to external examiner moderation as is common-place in UK HEIs' quality assurance procedures (Quality Assurance Agency for Higher Education, 2011). Whilst the scenario within the briefs changed across the four years (e.g. a class hierarchy of employees vs. a class hierarchy of shapes), the briefs themselves were consistent in that they required the student to design and implement a system based upon a class hierarchy, a container relationship and polymorphic message passing. The tagging grammar in Appendix A places no restriction upon the number of classes or relationships contained in an artefact and hence, within the confines of the object oriented context, the tool is scalable for more complex assignment briefs.

The submission data was randomly divided into test and development sets (see the table below). The development set was used to develop the tool and was not used in the evaluation. Figure 5.2 illustrates the initial use of a small number of student submissions to inform the tool's development with the number rising as the tool matured. This time-line reflects the evolutionary development of the tool and the need to ensure that sufficient submissions were left to undertake the planned evaluation.

The test set was not examined prior to its use in this evaluation. 168 student submissions were reserved for the test set. This number was determined by two factors. The first was that it needed to be a multiple of 12 as this was the number of experts who had agreed to participate in the evaluation (see section 5.4 below). The second was that there needed to be sufficient to send a sample of scripts to each expert.

Institution	Academic Session	No. of Student Submissions	No. used for development	No. reserved for evaluation
Institution A	2006-07	29	5	24
Institution A	2007-08	30	6	24
Institution A	2008-09	29	5	24
Institution B	2008-09	80	32	48
Institution B	2009-10(sem1)	23	11	12
Institution B	2009-10 (sem2)	59	23	36

Figure 5.2: The division of student submissions into developmental and evaluation sets.

5.4 Phase 1 Generating Feedback Data based on the Student Submission

Experts in the computer science education community were used in the evaluation of this research. Each expert marker was asked to mark a group of student submissions that were randomly allocated from the evaluative set. The expert markers were asked to produce both a summative grade and a set of formative feedback comments based upon the student submission.

Twelve human expert markers (academic staff members at 9 different UK HEIs) were recruited to grade and to provide feedback on the student submissions. The number of experts needed to be sufficient in order to avoid bias from one individual or from one part of the HEI sector. Twelve were chosen to achieve a broad range of views whilst keeping the number of people involved in the evaluation manageable. They were chosen to ensure representation from both the research-led (pre-92) and teaching-led (post-92 former polytechnics and Colleges of Higher Education) sectors. Grading used a pre-defined marking scheme to produce a summative percentage grade. The markers were instructed to write comments as they would normally provide to a student to reinforce and support the student's learning. No restriction was placed upon the number of comments that they could generate. After discussion with the team, each marker was given ten student submissions to mark. The feedback from the markers indicated that this was the greatest number we could expect them to return considering that they were employed full-time, their time on this project was additional and voluntary and this was the first of two evaluation activities that the team would be asked to participate in.

The main rationale for adopting this approach was that it provided a bank of feedback comments, generated by human experts, which could be used to compare against those comments generated by the tool. A secondary benefit was that, in marking a subset of the student work, the team became familiar with the context of the student submission and gained experience in generating feedback that they felt was both appropriate and which would usefully inform the student upon his/her learning. This familiarisation was important as the second phase of the evaluation asked them to compare and evaluate comments that were generated by both the tool and the other expert markers. Three members of the

marking team withdrew from the process citing pressures of work as the reason. The remaining nine members remained for the duration of the research project.

Of the 10 submissions sent to each marker, 3 were common to all markers and 7 were unique and distinct. The rationale for this choice was that having all markers grade the same (sub) set of student work enabled a check to take place for marking consistency within the team. The 3 common student scripts were chosen randomly from the evaluative set. Those remaining in the evaluative set were formed into groups with 7 student scripts allocated to each group. Each member of the evaluative team was then randomly allocated a group. The rationale for this was that a team member was allocated a set of student work from one year and from one institution. This minimised the number of assignment specifications that they needed to familiarise themselves with. Adopting this approach generated a summative grade and a set of feedback comments for 66 student scripts.

In summary, this section has discussed how a team of expert markers was used to generate a bank of both formative feedback comments and summative percentage grades as a consequence of marking the student submissions. The method and rationale used to randomly allocate student submissions to members of the team has been discussed. A (sub) set of assignments were chosen to be marked by all members of the team for the purpose of verification.

5.5 Testing for Consistency within the Team of Expert Markers

When assessing student work there can be variability in both the grades and formative comments generated by individual markers. There was a need to remove the comments from any team member who was viewing the student submission (statistically) significantly different to the others. Two statistical tests were undertaken to test for significant differences in the summative grades

produced by each marker. Section 1 of Appendix E provides the detail for both. As all markers returned their grades as a percentage mark, the first involved calculating the (population) mean mark and the standard deviation for each of the 3 marked common scripts. The following null hypothesis was postulated:-

$H_{0:}$ For this assignment, the mark generated by this individual team member is not significantly different to the marks generated by the marking team.

And the alternative hypothesis was:

$H_{a:}$ For this assignment, the mark generated by this individual team member is significantly different to the marks generated by the marking team.

As the population mean and standard deviation were known a two-sided, 95% confidence Z test was undertaken to test the null hypothesis (Diamond and Jeffries 2001).

The second test undertaken, advocated by Gwet (2010), was complementary to the method described above. The test takes advantage of the fact that all nine members of the team graded the same student submissions. They did so utilising a marking scheme that contains assessment grade criteria. This criteria specifies the features of the student submission required for the award of a grade A (excellent) through to E (fail). In this circumstance, Gwet (2010) advocates the use of the AC1 coefficient. This involves evaluating the extent to which two raters (expert markers) agree when they have analysed data and classified it into several non-overlapping categories. In this case, the raters classified the same 3 student scripts into the non-overlapping grades of A through to E. The AC1 coefficient was calculated and was used to measure the strength of agreement between the respective team members. The formative comments from two team members were

removed from the remainder of the evaluation as their grades were not statistically in agreement with the rest of the team**.**

The rationale for adopting two tests was that the Z test is one that is mature and established and thus offered the potential of a comparison to be made with other work. The AC1 coefficient is relatively new and consequently does not offer the same comparison potential but does however focus specifically on inter-rater reliability.

Upon completion of this process a set of formative feedback comments had been produced, generated by a consistent set of reviewers and based upon an analysis of the student work. Each new bank of comments could now be viewed holistically as if they were derived from a single population. They constituted a suite of representative formative feedback comments against which tool-generated comments could be compared.

5.6 Design of the Evaluative Questionnaires

Likert scales are widely used for measuring attitudes, opinions and preferences (Goeb *et al*. 2007). A typical example of such use that is now commonplace in the field of Higher Education in the United Kingdom is the National Student Survey. This survey presents final year undergraduate students with 22 statements. Each statement addresses aspects of the undergraduate educational experience. Participating students are asked to respond (positively or negatively) to each statement using a 5-point Likert scale.

The evaluation of this research centres on collating and evaluating informed opinion. Consequently, an integral component of the evaluation of this research is the adoption and use of a Likert scale. However, there are some statistical challenges associated with such an adoption. These include the following:

- The number of points on the scale.
- The format of the scale.
- Whether or not a mid-point should be included on the scale.
- Interpretation of Likert data

Appendix D discusses these challenges in detail and the rationale for adopting a 5-point Likert scale (and by implication the inclusion of a mid-point) with named points (*strongly agree, agree, neutral, disagree, strongly disagree)* and choosing the median and mode for describing and interpreting Likert data.

5.6.1 Questionnaire Used with the Evaluators

This section presents the questionnaire that was used to survey the team of evaluators with regard to evaluating the formative feedback comments generated by the tool. The three categories of quality, relevance and coverage that the team were invited to rate the comments against are introduced and defined. The Likert scale adopted in the evaluation is specified.

The team of evaluators were presented with a set of comments and asked to rate them. The comments came from a sample that included both those that were generated by the team of expert markers and those that were generated by the tool. This meant identifying suitable criteria against which the comments would be rated. This is a substantial problem because, for example, giving students detailed and comprehensive feedback may not be a good policy as a long list of issues that need to be addressed may not always be read and acted upon. There is a need to strike a balance between issues that represent a misunderstanding of the main learning outcomes being assessed and those that are relatively minor or tangential to the aims of the assignment brief.

Fourteen evaluative statements were designed against which evaluators would be asked to rate the comments. The statements were derived from considering three broad criteria of :

- Quality
- Relevance
- Coverage

Quality is concerned with the extent to which the comments adequately described the item being fed back upon and the extent to which it provided support to the students for their learning.

Relevance is concerned with, for the student submission as a whole, the priority given to feeding back on one particular issue at the expense or even omission of another. For example, it is possible to imagine that good quality feedback that is focused in one area of the submission is at the expense of generating feedback of a similar quality in another. The issue of relevancy applies when this second area relates to a fundamental error in the student submission or is crucial to supporting the student's learning.

Coverage is concerned with ensuring that feedback comments are generated across the spectrum of all issues of relevance contained in the student submission.

Each team member was presented with 14 evaluative statements. The team member was asked to judge the feedback comments against these statements utilising a Likert scale. The Likert scale adopted was a 5-point named scale for all three sets of statements. Five-point was chosen because of its reliability over scales with fewer points (McKelvie 1978). It also contained a mid-point to minimise results that may be misleading (Matel and Jacoby 1972). The same five point scale was used for all statements to ensure consistency of responses by the team (Norvell 1977). The names of the scale were similar to those observed by Goeb *et al.* (2007) as being common-place for such scales:

- Strongly Agree
- Agree
- Neither Agree nor Disagree
- Disagree
- Strongly Disagree

The 14 statements that each team member was asked to consider for each set of feedback comments they were asked to review were:-

Quality

1) The majority of comments contained in this set are clear.
2) The majority of comments contained in this set are concise.
3) The set of comments provide sufficient detail in order for a student to know what concept or issue is being fed back upon.
4) The set of comments contained in this set provide sufficient detail in order for a student to know what further work they need to undertake.
5) The set of comments will help the student with his/her learning.

Relevance

6) The comments contained in this set are relevant for this type of assignment brief and the associated indicative learning outcomes.
7) The comments contained in this set address important areas of strength found in the student submission that is considered to be of significance.
8) The comments contained in this set address important areas of weakness found in the student submission that is considered to be of significance.
9) It is clear which concepts the comments in this set are addressing.
10) The comments in this set will help the student with his/her learning.

Coverage

11) This set of comments, when viewed in its entirety, fully encapsulates **all** pertinent feedback needed for the student to recognise where there are areas of strength in the submission.

12) This set of comments, when viewed in its entirety, fully encapsulates **all** pertinent feedback needed for the student to recognise where there are areas of weakness in the submission and where further learning is required.

13) This set of comments would provide a useful enhancement to the type of comments that I gave during stage 1 of this evaluation.

14) This set of comments would have been sufficient to replace the type of comments that I gave during stage 1 of this evaluation.

Each question was designed to be as unambiguous as possible, covering a single idea, in an attempt to ensure consistent interpretation across all members of the evaluative team. The questions were comprehensively reviewed by a group of academic colleagues and a set of guidance notes were produced and sent to each member of the team (see Appendix C).

This section presented the questionnaire that was used to survey the team of evaluators with regard to evaluating the tool- and human-generated formative feedback comments. The 14 statements that the team were asked to rate the comments against were introduced and their grouping into the three categories of quality, relevance and coverage was highlighted. The questionnaire that was sent to the team of evaluators was discussed and the Likert scale adopted in the evaluation was specified. Appendix C presents the questionnaire together with the explanatory notes given to the team of evaluators.

5.6.2 Questionnaire Used with the Student Body

This section presents the questionnaire that was used to solicit the student view in the evaluation of this research.

Each student was presented with a set of feedback statements that had been generated by the tool as a consequence of analysing the work submitted. The work consisted of a design diagram and an accompanying implementation. The students were asked to rate the feedback comments generated by the tool against a set of statements utilising the same 5-point Likert scale used with the evaluators.

The statements were informed by the literature discussed in Chapter 2 relating to the use of the student body to evaluate tools that had been developed to automate the assessment of diagrams. Consequently, the following questionnaire was used and was derived from an amalgamation of those used by Higgins *et al.* (2009) and Tselonis (2008).

- The feedback presented to me is helpful.
- The feedback presented to me is clear.
- The feedback presented to me is relevant to my solution.
- It is clear to me what concept the feedback is addressing.
- The feedback presented to me will help me to improve my solution.
- I will use this feedback to research further into this topic area.
- The feedback has helped me identify the strengths of my submission.
- The feedback has helped me identify the weaknesses contained in my submission.
- The feedback represents a useful enhancement to that which I received from my tutor.
- The feedback I received is sufficient enough for it to replace that which I received from my tutor.

5.7 Phase 2 Ensuring Consistency between Evaluators

Returning to the evaluators' experiment, phase 1 produced a set of summative grades and feedback comments from a (randomly chosen) sample of 66 student scripts. These were generated by the team of 9 expert markers. The summative grades from (randomly chosen) three of these scripts were used to check that the expert markers were consistent with each other in their marking. The remaining 63 sets were collated into a bank. Those comments generated by the tool were also collated and stored in a separate bank. Both banks were used in phase 2 of the evaluation.

Phase 2 consisted of distributing the feedback comments among the evaluators for review. Evaluation was conducted by asking each evaluator to complete the questionnaire for each set of comments received. This section describes how the comments were distributed among the evaluators and the statistical analysis undertaken to ensure that the evaluators were viewing the formative comments consistently between each other. Section 5.8 discusses the distribution and those tests taken in order to compare ratings given to the tool-based comments with those for the comments generated by the team of expert markers.

It is not necessarily the case that an individual expert marker who marked the student scripts significantly differently from the rest of the team would also have reviewed formative feedback comments differently (and vice versa). Hence, there needed to be a test in phase 2 to ensure consistency within the team when evaluating feedback comments. Consequently, the tool was applied to three (randomly chosen) student submissions and the feedback generated was collated. Each team member was sent all three sets of feedback to evaluate and asked to complete the questionnaire.

Gwet's (2010]) AC2 coefficient was used to analyse the inter-rater reliability between the evaluators. The AC2 coefficient is an extension to AC1 to address

data that is ordinal. Likert data is ordinal and hence, for this phase of the evaluation, the AC2 coefficient was preferable to AC1. Section 2 of Appendix E discusses the AC2 coefficient in detail.

There was consistency in the team when rating feedback comments against 13 of the 14 questionnaire statements. Hence, it was these 13 statements that were used to compare ratings between tool- and human-generated comments. There was no consensus for statement 2 (the issue of conciseness) and hence this was not used in the comparison.

In summary, this section has discussed the steps taken to militate against significant differences in the individual members of the reviewing team when rating formative feedback comments. It described how the comments generated by the tool for 3 randomly chosen scripts were utilised to do this. It described the rationale for choosing the statistical technique undertaken to test for significant differences between individual members of the team of evaluators.

5.8 The Allocation and Evaluation of Comments by the Evaluative Team

The 63 sets of comments generated by the expert markers were used in the evaluation of the tool. This section describes the distribution of comments to members of the team of evaluators and the statistical tests employed to compare the feedback comments generated by the tool with those generated by the team of expert markers.

Phase 2 consisted of asking a team of evaluators to evaluate a sample of comments. The sample was randomly generated from both banks of comments ensuring that:

1. No member of the evaluative team was asked to evaluate comments that they themselves had generated.
2. Each member of the team was asked to evaluate a (sub)set of comments generated by the tool in addition to a (sub)set of comments generated by the evaluative team.
3. The evaluative team were were not told which comments had been produced by the tool and which had been generated by the team.

Each team member's evaluation was undertaken separately and independently to the other members. Each member was sent 10 sets of comments to evaluate. Each member was asked to complete 10 questionnaires, one for each of the 10 sets of comments.

Figure 5.3 shows how comments from both banks of comments were (randomly) distributed amongst the 9 team members. It also shows how each team member was sent the same 3 sets of comments to enable a statistical test to be undertaken to check for variations between individual team members as discussed in section 5.7 above.

The completed questionnaires were used in the evaluation of the tool-generated comments. For each of the 13 statements in the questionnaire, the returns from the evaluators produced 32 Likert scores for tool-generated comments and 31 for those that were human-generated. Consequently, for each statement, it was possible to calculate the population median Likert score and to consider the 32 Likert scores for the tool-generated comments to be a sample taken from a population of 63 scores in total.

Evaluator reference	No. of sets of comments taken from the tool bank	No. of sets of comments taken from the team bank	No of sets of comments that were common to all markers
Marker2	4	3	3
Marker3	3	4	3
Marker4	4	3	3
Marker5	3	4	3
Marker6	4	3	3
Marker7	3	4	3
Marker8	4	3	3
Marker9	3	4	3
Marker10	4	3	3

Figure 5.3: A table to indicate the allocation of tool and team based comments to members of the evaluative team.

Two statistical tests were performed to compare the evaluations of the human- and tool-generated comments. These were a sign test and a Mann-Whitney U test. They were chosen because:

- They are standard non-parametric tests suitable for the ordinal nature of Likert data.
- Being standard tests they enable a comparison to be made with existing or future surveys in this field.
- Each test analyses the data differently. For example, the sign test compares the median of a sample with the population median whereas Mann Whitney tests for differences between two different groups. Applying

the two tests provides a richer description of the data and the inferences that can be drawn.

Both tests considered whether or not there was any statistically significant difference between the Likert scores generated for the tool and human-generated comments. The tests were applied to each one of the 13 statements in the questionnaire.

The non-parametric one-sample sign test was used to compare the median Likert score for the tool-generated comments with the median Likert score for the population. The null hypothesis for the test was

H_0 – The Likert scores for the tool-generated comments are distributed such that half of them lie above the population median.

And the alternative hypothesis was

H_a – The Likert scores for the tool-generated comments are distributed such that half of them do not lie above the population median.

A two-sided Mann-Whitney U test was used to test if the distribution of Likert scores for the tool-generated and human-generated comments were the same. The null hypothesis was

H_0 – There is no difference in the distribution of Likert scores between the tool- and human-generated comments.

And the alternative hypothesis was

H_a – There is a difference in the distribution of Likerts scores between the tool- and human-generated comments.

Where a statistically significant difference was detected there was a need to describe the direction of the difference (i.e. were the tool-generated comments rated higher than human-generated or vice-versa). The direction was determined

by analysing the median score for each group. This approach is advocated by Pallant (2007). An alternative method would have been to conduct a one-sided test with the alternative hypothesis being that the tool-generated tools had lower scores than those that were human-generated.

In summary this section described how a team of evaluators were sent a random allocation of feedback comments to evaluate. Each allocation consisted of both tool- and human-generated comments. Evaluation took place through the use of a questionnaire. Two statistical tests were made on the returns. Each test compared the evaluations of the tool-generated comments with those that were human-generated. The results of the tests are analysed in the next chapter.

5.9 Evaluation by the Student Body

Feedback produced by the tool will ultimately be read by the student. Hence, the students' view was sought on whether the tool's comments would be helpful with their learning. This section describes how the student view was sought and collated. It describes how a questionnaire was used to rate the feedback that was generated by the tool as a consequence of analysing the coursework that the students submitted.

Semester 2 of academic session 2010-11 saw 30 students submit a UML design diagram with an accompanying implementation. The tool was applied to a subset of this submission and the formative feedback comments generated were collated. The students were asked to complete the evaluation questionnaire as discussed in section 5.6.2. They were completed in class via an Electronic Voting System (EVS). This offered the advantage of a timely collation of evaluations whilst maintaining student anonymity. The completed questionnaires were collated and an analysis of the results is presented in Chapter 6.

5.10 Summary

Student submissions were gathered over four years. An evaluative team marked them producing a summative grade and a set of feedback comments. Comments were discarded from team members whose summative grades were significantly different. The remaining comments were combined with those produced by the tool and the team were asked to evaluate them. No team member was asked to evaluate their own comments. Team members were not told which were tool and which were the human-generated comments.

Evaluation was performed via completing a questionnaire. The questionnaire contained 14 statements against which comments were evaluated on a 5-point Likert scale. The 14 statements focused on the categories of quality, coverage and relevance. Returns from members whose evaluations were significantly different to the rest of the team were discarded.

Three statistical experiments were conducted. The first used a Z test and the AC1 coefficient to test for significant differences in summative grades. The second used the AC2 coefficient to test for significant differences within the team in evaluating feedback comments. The third used a sign test and a Mann-Whitney U test to compare tool- with human-generated comments.

A further questionnaire was used to obtain the students' view of the tool-generated comments. It consisted of 10 statements against which the comments were evaluated using a 5-point Likert scale. The questionnaire was conducted in-class using an Electronic Voting System.

The results of this experimentation are analysed and reported upon in Chapter 6.

Chapter 6. Results

6.1 Introduction

The previous chapters have discussed the development of a proof-of-concept assessment tool that automatically generated formative feedback based upon an analysis of a design diagram and its accompanying implementation. The methodology adopted to evaluate the feedback generated by the tool was presented. This chapter presents the results of the evaluation. It is structured in four sections. The first three focus upon an evaluation undertaken by members of the academic subject community, whilst the fourth focuses upon the student evaluation. In particular, Section 6.2 presents the results obtained in an experiment to test if human markers were grading the student work comparably. Section 6.3 presents the results obtained in an experiment to test if the team of human evaluators were rating formative feedback comments comparably. Sections 6.4 and 6.5 present the evaluators' and students' respective evaluation of the tool-generated feedback comments. The final section provides a conclusion and summary of the analysis of these results and a reflection upon the evaluative process.

6.2 Consistency in the Marking Team and the Collation of Human-Generated Feedback Comments

Of the twelve members of the academic community who agreed to participate in this research three members withdrew (markers 1, 11 and 12) in the early stages citing pressures at work. The remaining nine members engaged with the research project through to completion. Each member was sent a randomly allocated set of 7 distinct student submissions taken from the evaluative set. Additionally each member was sent the same three student submissions also taken randomly from the evaluative set (labelled ass17, 79 and 182). Members were asked to generate

a summative percentage mark and a set of formative feedback comments for each student submission that they received. This section focuses upon the summative mark for the three common student submissions,

The summative mark received for each script was a percentage grade. This grade was used to test whether or not each member was viewing the student submission similarly. The three common scripts all scored very highly, making them less useful for discrimination between markers. One marker omitted to return a grade for one submission. Two statistical tests were undertaken: Gwet's (2010) multiple rater AC1 coefficient for inter-rater reliability and a Z test applied to each member of the marking team.

The AC1 coefficient is a statistical measure of how raters agree when they categorise items. In order to generate an AC1 coefficient for the (percentage) grades received they were tabulated into the following alpha grades:-

Percentage Mark	Alpha Grade
70%<=mark<=100%	A
60% <= mark< 70%	B
50% <= mark< 60%	C
40% <= mark< 50%	D
0% <= mark< 40%	E

Table 6.0 Mapping of Percentage Marks to Alpha Grades

The result of this classification is shown in Figure 6.1. The AC1 coefficient for this data is 0.84. Gwet (2010) notes that there are several existing benchmarks from which the strength of agreement between raters can be made. The benchmarking result for these coefficients indicates that the strength of agreement between

raters is “substantial” to “almost perfect” (Landis and Koch (1977)), “excellent” (Fleiss (1981)) and “good” to “very good” Altman (1991).

However, further analysis of the original percentage marks showed a variation within the ‘A’ alpha grade category (for example, 75% to 95% in ass182). A Ztest, was therefore undertaken on the summative marks returned for each of the student submissions. The results are tabulated on the table in Figure 6.2. Taking a Z-score that lies outside the range of -1.96 to 1.96 (95% confidence interval) as significant, marker 8 for assignment 17 and marker 3 for assignment 79 viewed the student submission significantly differently to the rest of the markers (a further ANOVA test, reported in appendix F, was undertaken identifying the same markers as being inconsistent). Consequently, the returns from these two markers, both grades and formative feedback comments, were removed from the remainder of the evaluation. This meant that the set of human-generated comments that formed the evaluative set for the remainder of the evaluation came from the markers 2,4,5,6,7,9 and 10. The AC1 test did not pick up on the two erratic markers because the differences in percentage marks were primarily subsumed within a large (70%<= mark<=100%) alpha grade. In retrospect, a finer granularity within the ‘A’ alpha grade category (for example A-, A, A+) could have been adopted. However, this would not have reflected current grading practice within the marking team and hence adopting a finer granularity was rejected.

In summary, Gwet’s (2010) AC1 coefficient indicated that when viewing the returns holistically there was a very good correlation within the marking team. However, a Z test showed that two markers were viewing some assignments significantly differently to the others. Consequently, their formative comments were removed from the remaining stages of the evaluation.

AC1 -Coefficient

	A	B	C	D	E	Omissions*		Total
ass 17	9	0	0	0	0	0		9
ass 79	7	1	0	0	0	1		9
ass 182	8	1	0	0	0	0		9
Average	8	1	0	0	0	0		

Inter-rater Reliability Coefficients, and Associated Standard Errors – excluding the omissions* category

		Inference/Subjects	
Method	Coefficient	StdErr	95% C.I.
Gwet's AC1	0.836671	0.084911	0.471 to 1

* Omissions refers to the case where one marker failed to return a summative mark for one of the assignments

Figure 6.1 Table to show how the data was modelled to generate the AC1 coefficient for Inter-rater reliability. 9 raters , 3 cases (assignments) and 5 categories (grades A to E).

Ref	Ass 17 %	Ass79 %	Ass182 %	Marker Average %	Zscore Ass 17	include comments based on Ass17	Zscore Ass79	include comments based on Ass79	Zscore Ass 182	include comments based on Ass 182	Include comments based on all three assignments
Marker 1	NA										
Marker 2	79	82	85	82.00	-0.70	Y	0.08	y	0.05	Y	y
marker 3	75	60	75	70.00	-1.51	Y	-2.21	n	-1.07	Y	n
marker 4	82	86	91	86.33	-0.09	Y	0.50	y	0.72	Y	y
marker 5	82	No return	68	75.00	-0.09	Y	No return	na	-1.85	Y	y
marker 6	80	73	81	78.00	-0.50	Y	-0.86	y	-0.40	Y	y
marker 7	84	92	93	89.67	0.32	Y	1.12	y	0.94	Y	y
marker 8	94	89	94	92.33	2.35	N	0.80	y	1.06	Y	n
marker 9	85	82	79	82.00	0.52	Y	0.08	y	-0.62	Y	y
marker 10	81	86	95	87.33	-0.29	Y	0.50	y	1.17	Y	y
Marker 11	NA										
Marker 12	NA										
Assignment Mean	82.44	81.25	84.56								
Assignment POP STD DEV	4.92	9.63	8.95								
Zscore sum					0.00		0.00		0.00		

Figure 6.2 Table to show the Ztest results for the percentage grades received for three, randomly chosen, student submissions

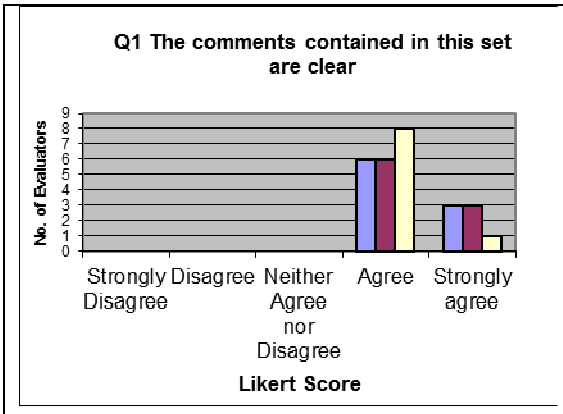
6.3 Consistency within the Team When Evaluating Feedback Comments

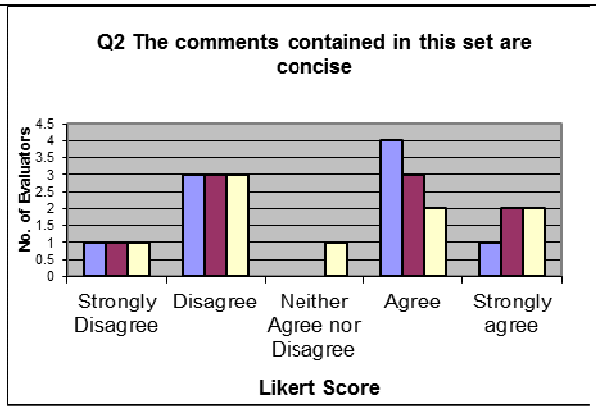
This section addresses the issue of ensuring that the team of evaluators were consistent when rating formative feedback comments. Three sets of tool-generated feedback comments (labelled feedback64, feedback65 and feedback66), taken from the evaluative set, were sent to each team member. Each member was asked to complete a questionnaire for each set of comments. The questions and the design and form of the questionnaire were described in Chapter 5, Section 5.6.1. The raw data is presented in Figure 6.3 and the tabulated results are presented in Figure 6.4.

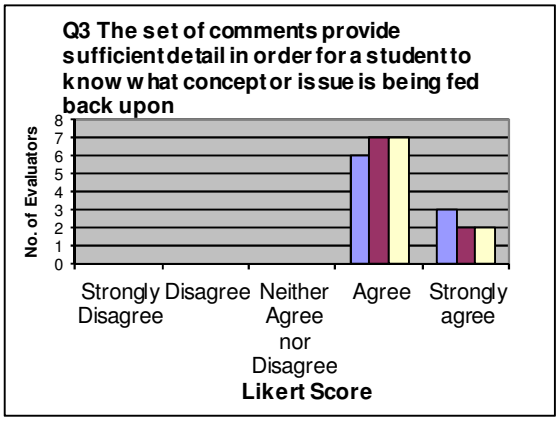
Q1 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	0	0	6	3
feedback 65	0	0	0	6	3
feedback 66	0	0	0	8	1
Q2 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	1	3	0	4	1
feedback 65	1	3	0	3	2
feedback 66	1	3	1	2	2
Q3 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	0	0	6	3
feedback 65	0	0	0	7	2
feedback 66	0	0	0	7	2
Q4 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	0	0	7	2
feedback 65	0	0	0	7	2
feedback 66	0	1	0	7	1
Q5 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	0	0	6	3
feedback 65	0	0	0	5	4
feedback 66	0	0	0	7	2
Q6 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	0	0	6	3
feedback 65	0	0	0	5	4
feedback 66	0	0	0	7	2
Q7 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	0	0	6	3
feedback 65	0	0	0	4	5
feedback 66	0	0	1	2	6

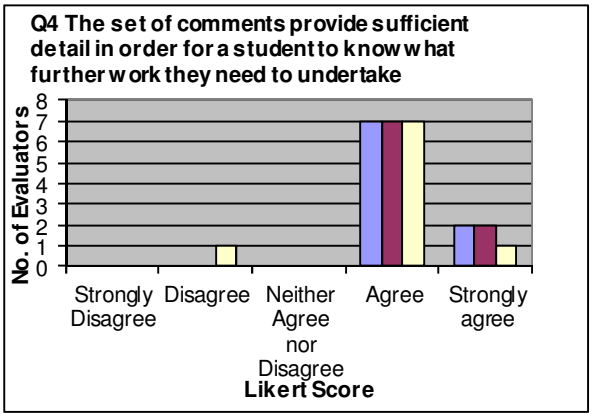
Q8 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	0	0	5	4
feedback 65	0	0	0	4	5
feedback 66	0	1	0	3	5
Q9 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	0	0	6	3
feedback 65	0	0	0	5	4
feedback 66	0	0	0	6	3
Q10 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	0	0	6	3
feedback 65	0	0	0	5	4
feedback 66	0	0	0	5	4
Q11 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	1	0	6	2
feedback 65	0	0	1	5	3
feedback 66	0	1	1	5	2
Q12 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	0	1	4	4
feedback 65	0	0	1	4	4
feedback 66	0	1	1	3	4
Q13 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	0	0	4	5
feedback 65	0	0	0	5	4
feedback 66	0	0	0	4	5
Q14 inter-rater analysis					
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly agree
feedback 64	0	0	3	2	4
feedback 65	0	0	3	2	4
feedback 66	0	1	3	1	4

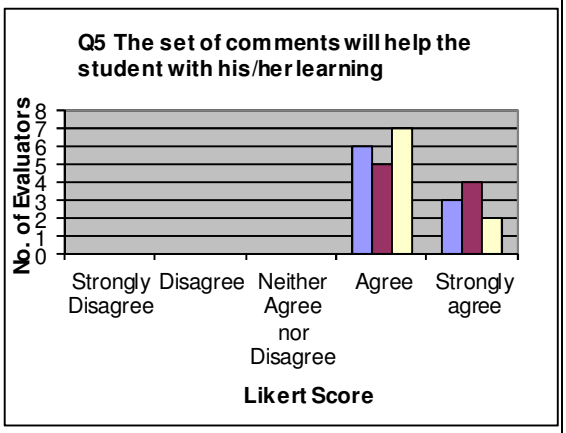
Figure 6.3 Table of the raw Likert data returns for the three common scripts

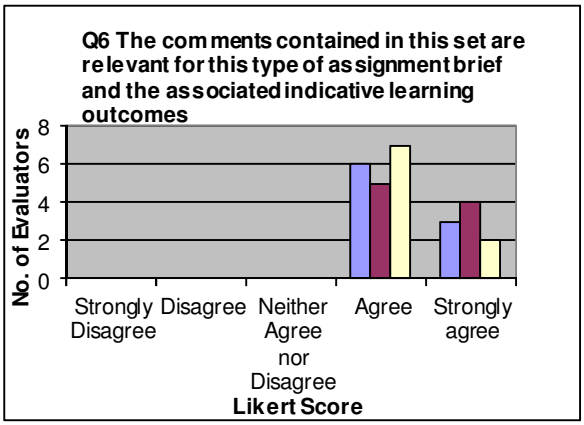

Q1 The comments contained in this set are clear
No. of Evaluators
0
1
2
3
4
5
6
7
8
9
Strongly Disagree
Disagree
Neither Agree nor Disagree
Agree
Strongly agree
Likert Score

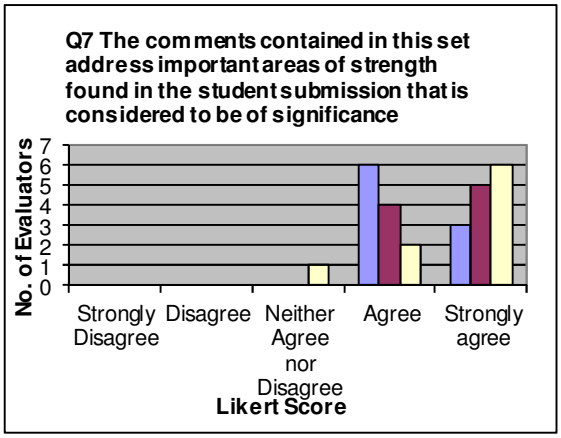

Q2 The comments contained in this set are concise
No. of Evaluators
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
Strongly Disagree
Disagree
Neither Agree nor Disagree
Agree
Strongly agree
Likert Score

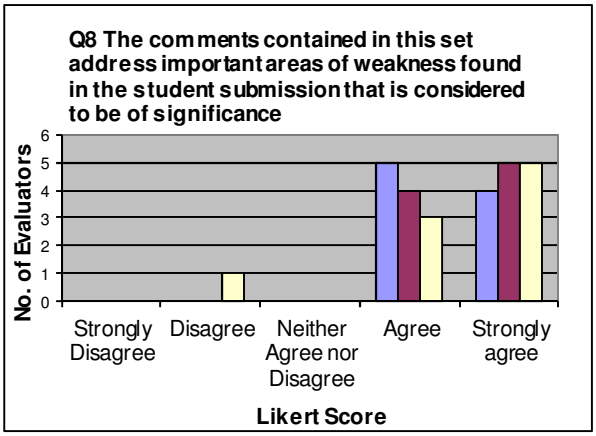

Q3 The set of comments provide sufficient detail in order for a student to know what concept or issue is being fed back upon
No. of Evaluators
0
1
2
3
4
5
6
7
8
Strongly Disagree
Disagree
Neither Agree nor Disagree
Agree
Strongly agree
Likert Score


Q4 The set of comments provide sufficient detail in order for a student to know what further work they need to undertake
No. of Evaluators
0
1
2
3
4
5
6
7
8
Strongly Disagree
Disagree
Neither Agree nor Disagree
Agree
Strongly agree
Likert Score


Q5 The set of comments will help the student with his/her learning
No. of Evaluators
0
1
2
3
4
5
6
7
8
Strongly Disagree
Disagree
Neither Agree nor Disagree
Agree
Strongly agree
Likert Score


Q6 The comments contained in this set are relevant for this type of assignment brief and the associated indicative learning outcomes
No. of Evaluators
0
2
4
6
8
Strongly Disagree
Disagree
Neither Agree nor Disagree
Agree
Strongly agree
Likert Score


Q7 The comments contained in this set address important areas of strength found in the student submission that is considered to be of significance
No. of Evaluators
0
1
2
3
4
5
6
7
Strongly Disagree
Disagree
Neither Agree nor Disagree
Agree
Strongly agree
Likert Score


Q8 The comments contained in this set address important areas of weakness found in the student submission that is considered to be of significance
No. of Evaluators
0
1
2
3
4
5
6
Strongly Disagree
Disagree
Neither Agree nor Disagree
Agree
Strongly agree
Likert Score

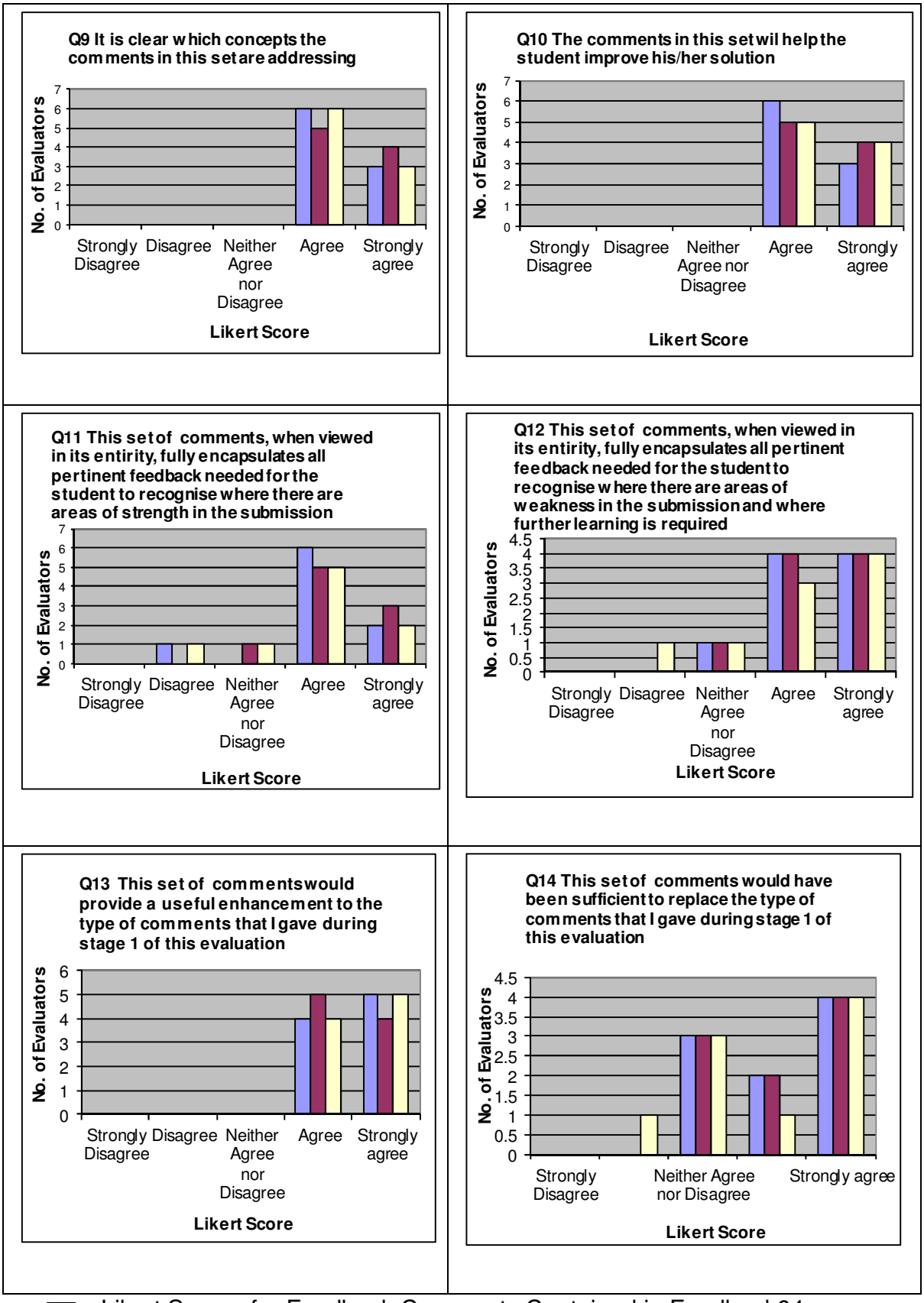


Figure 6.4 Table to illustrate the questionnaire returns used to evaluate the extent of inter-rater consistency within the evaluative team.

An analysis of this data shows that the evaluators uniformly 'agreed' or 'strongly agreed' with 7 of the 14 statements contained in the questionnaire (statements 1,3,5,6,9,10 and 13). This increases to 8 from 14 if the neutral 'neither agree nor disagree' response is included (statement 7). Gwet's (2010) AC2 inter-rater reliability coefficient for multiple raters is tabulated for each statement contained in the questionnaire in Figure 6.5. This indicates that, with the exception of statement 2, consistency within the team was "almost perfect" (statements 1,3,4,5,6,7,8,9,10) or "substantial" (statements 11,12,13,14) – (Landis and Koch scale -1977).

On the criterion of quality (statements 1,2,3,4,5) and relevance (statements 6,7,8,9,10) evaluators rated the comments consistently, with the majority of the returns confined to the agree/strongly agree ratings. The consistency of the evaluators, whilst being "substantial", was not as strong for the statements associated with the criterion of coverage. Evaluators were clearly challenged when asked to consider whether or not the comments they were evaluating could replace the type of comments they themselves produced (statement 14). This statement produced the most returns in the neutral neither agree nor disagree category.

	Gwet AC2	Agreement Benchmark (Landis and Koch - 1977)
Q1	0.96	Almost Perfect
Q2	0.19	Slight
Q3	0.96	Almost Perfect
Q4	0.93	Almost Perfect
Q5	0.95	Almost Perfect
Q6	0.95	Almost Perfect
Q7	0.91	Almost Perfect
Q8	0.87	Almost Perfect
Q9	0.94	Almost Perfect
Q10	0.93	Almost Perfect
Q11	0.73	Substantial
Q12	0.78	Substantial
Q13	0.78	Substantial
Q14	0.66	Substantial

Figure 6.5 Gwet's (2010) AC2 Inter-rater reliability coefficient for the 3 common scripts

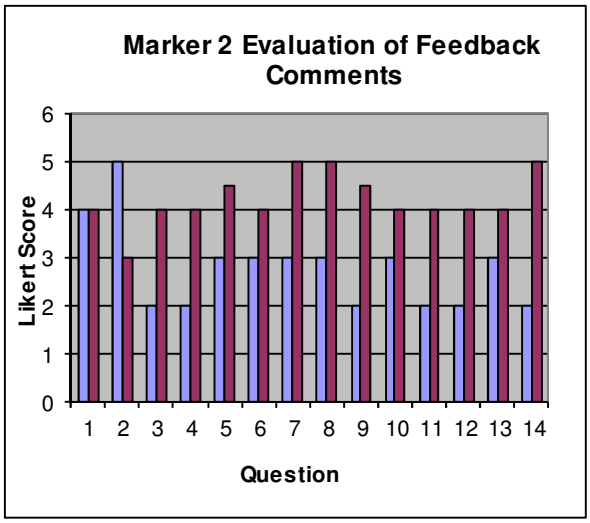
The issue of conciseness (statement 2) was the statement for which there was the most disagreement within the evaluators with the data being spread across the full spectrum of 'strongly disagree' to 'strongly agree' categories. Of the 27 returns (9 evaluators, 3 sets of comments evaluated by each evaluator) 14 evaluators 'agreed' or 'strongly agreed' with the statement "The comments contained in this set are concise" whilst 12 'disagreed' or 'strongly disagreed' (one return was neutral). This has implications for interpreting the evaluation of comments generated by the tool on the conciseness criterion. If the evaluators could not agree upon rating a feedback comment then it is not sound to infer anything conclusive about their view on this criterion for the comments generated by the tool. This is reflected upon further in chapter 7.

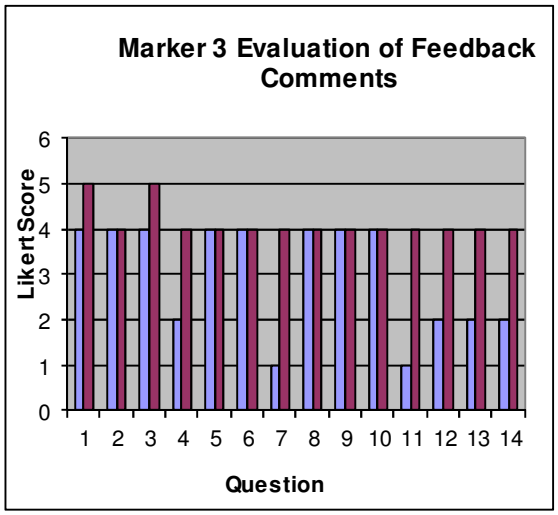
In conclusion, there was consistency in the team when rating feedback comments for 13 of the 14 questionnaire statements. Hence, it was these 13 statements that were used to compare ratings between tool- and human-generated comments. There was no consensus for statement 2 (the issue of conciseness) and hence this was not used in the comparison.

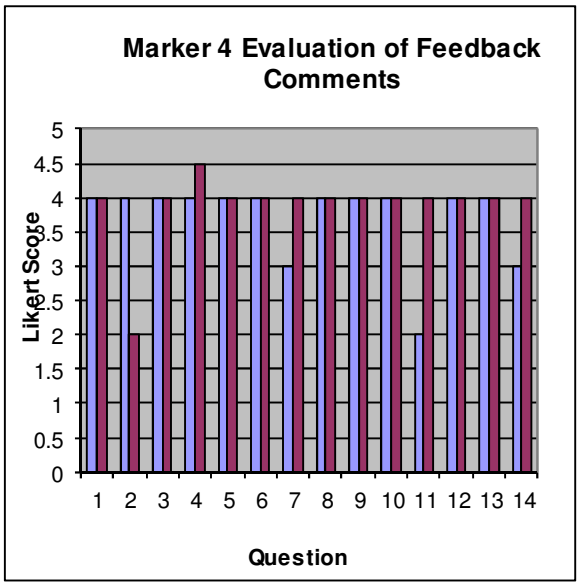
6.4 An evaluation of Tool -Generated Comments Compared with Human-Generated.

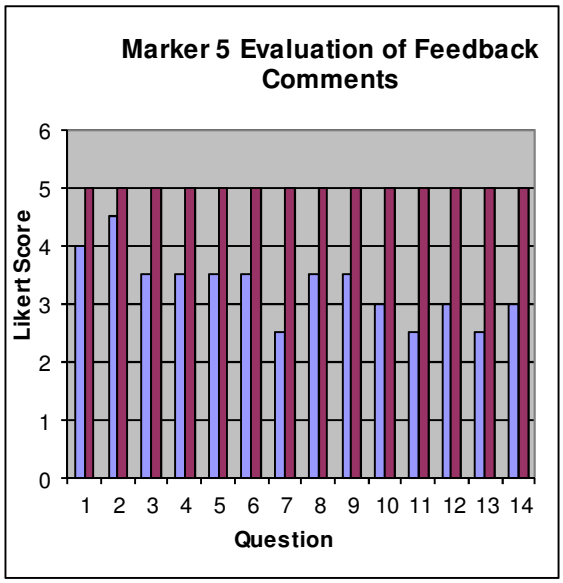
The previous section concluded that members of the evaluative team were viewing and rating formative comments very similarly. This section focuses upon the comparison between feedback comments generated by the tool with those generated by the team of expert markers. Nine evaluators were sent seven distinct sets of comments. Each set contained a random allocation of both human- and tool-generated comments. Each set consisted of either 4 tool-generated and 3 human-generated sets of comments or vice-versa. This resulted in 63 questionnaire returns of which 31 evaluated formative comments that were human generated and 32 that were tool generated. This section presents the results and an analysis of these returns. The previous chapter discussed the ordinal nature of Likert data and the inappropriateness of using the mean Likert score in this context. Hence, the results are presented using median and mode Likert scores.

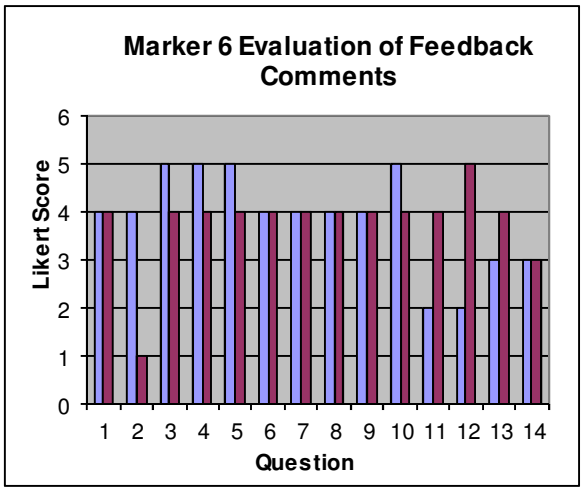
Figure 6.6 below shows the median Likert scores returned by each evaluator for all the statements contained in the questionnaire. The medians for the human and tool generated comments were calculated separately. Figure 6.7 shows a similar tabulation based upon the modal Likert scores. Where the data resulted in two (bimodal) or more (multimodal) modes, a value of zero was prescribed (example Question 5 and 6 for marker 2).

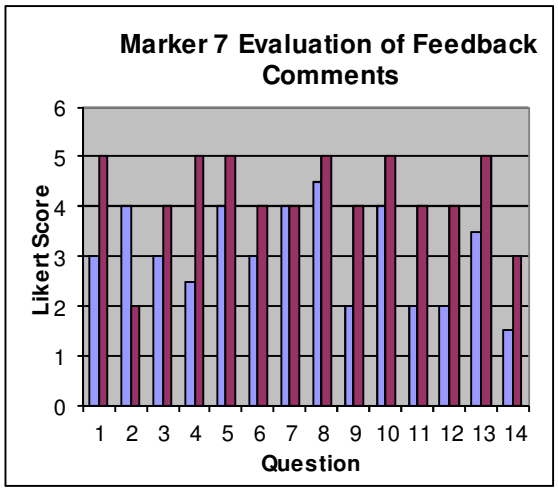

Marker 2 Evaluation of Feedback Comments
Likert Score
0 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Question


Marker 3 Evaluation of Feedback Comments
Likert Score
0 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Question


Marker 4 Evaluation of Feedback Comments
Likert Score
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Question


Marker 5 Evaluation of Feedback Comments
Likert Score
0 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Question


Marker 6 Evaluation of Feedback Comments
Likert Score
0 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Question


Marker 7 Evaluation of Feedback Comments
Likert Score
0 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Question

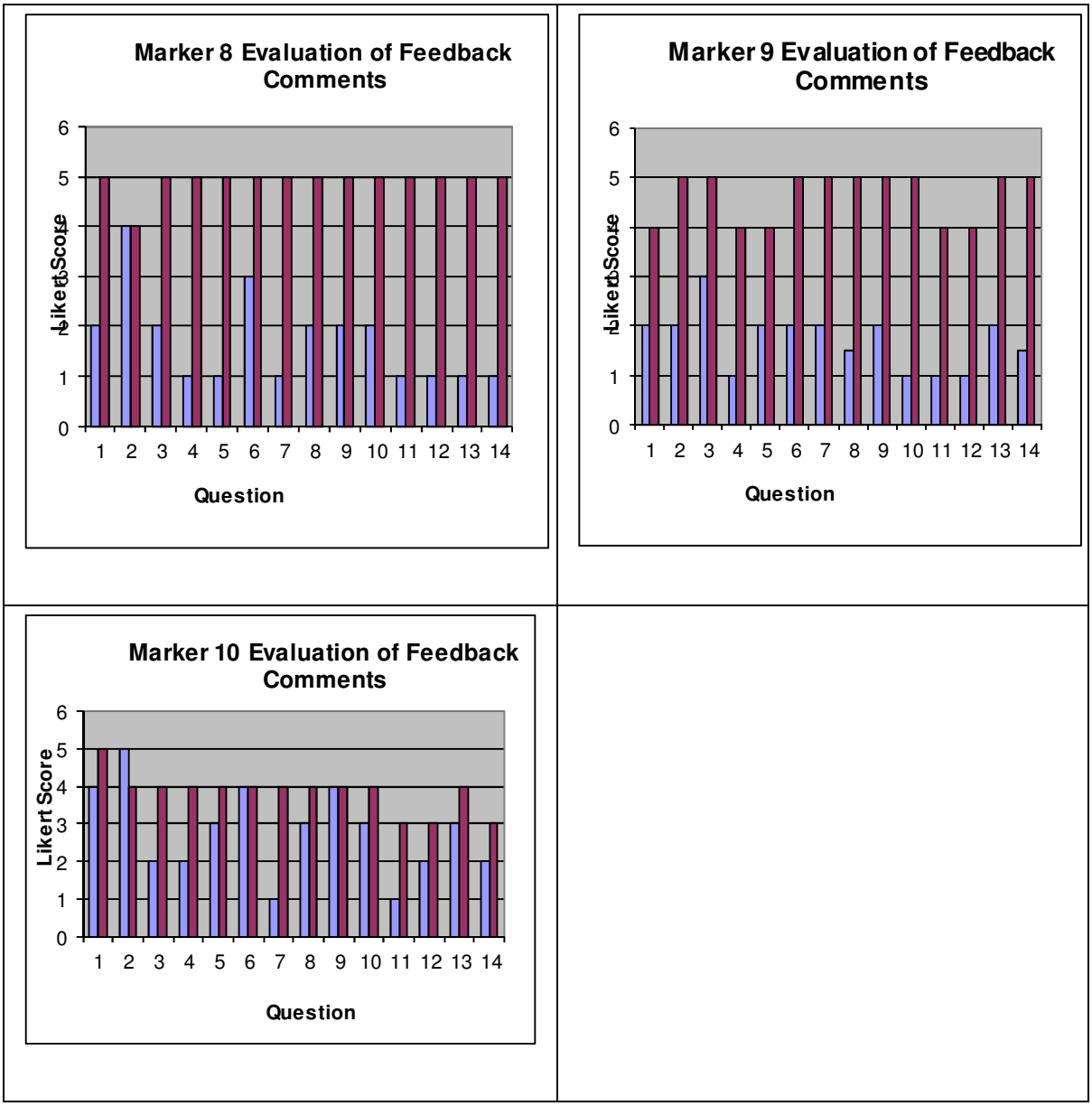
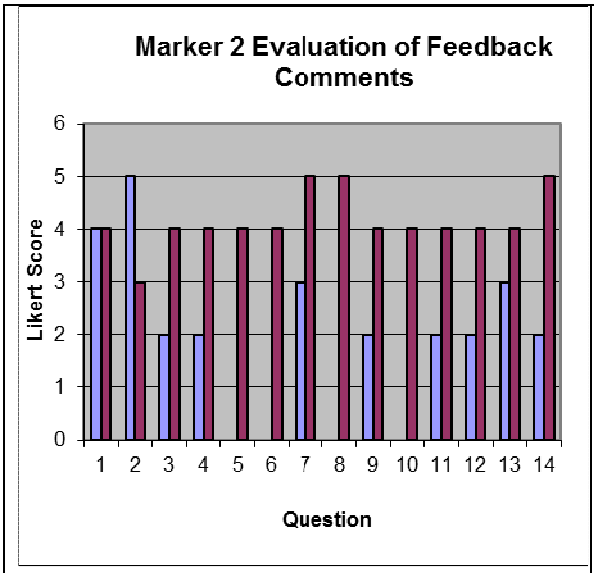
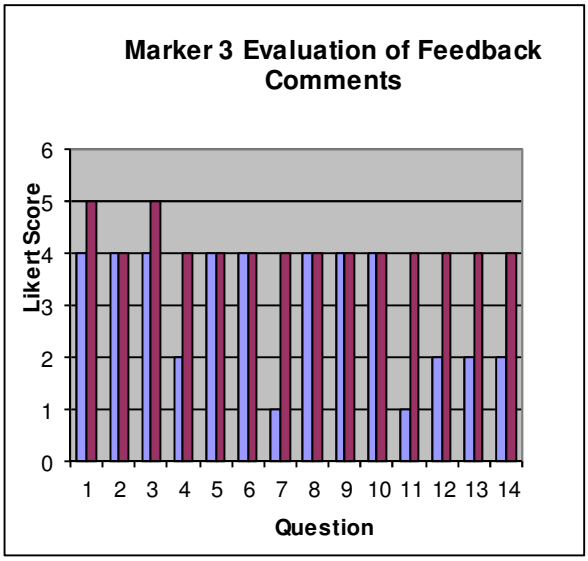
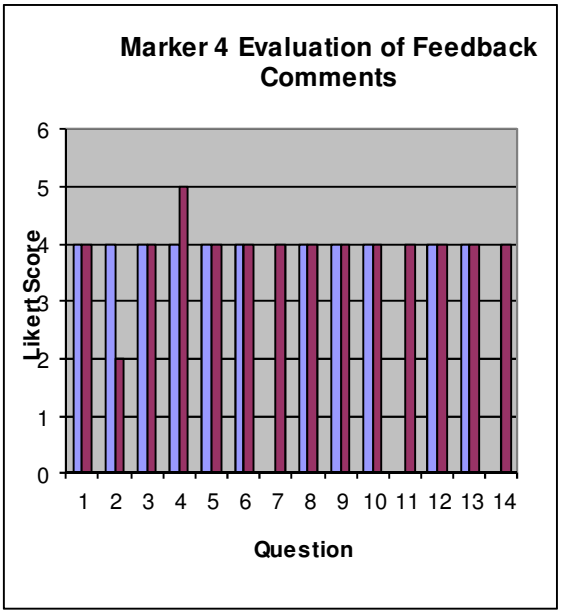
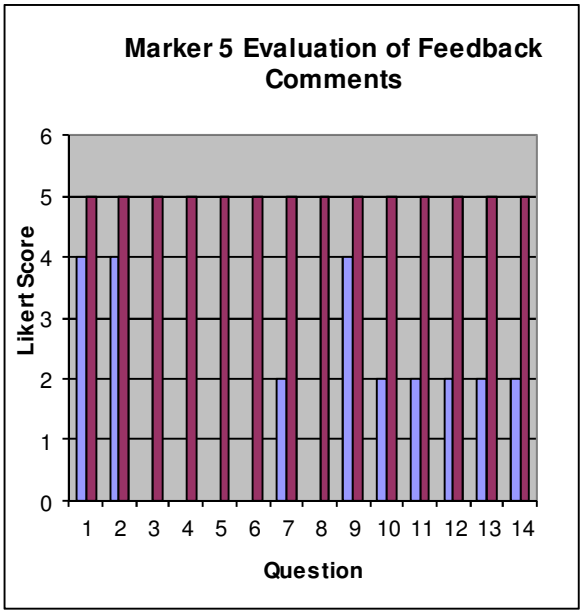


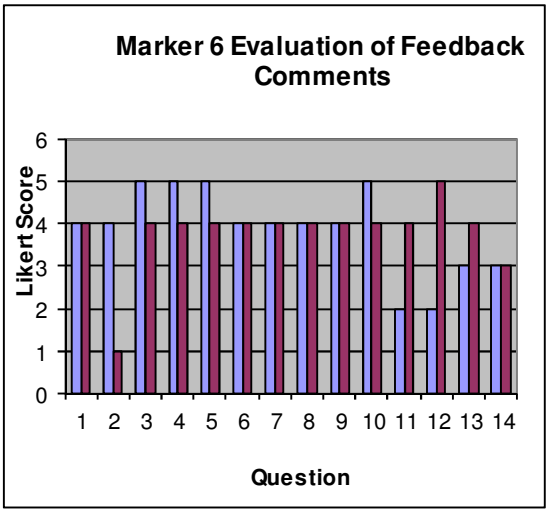
Figure 6.6 Median Likert Scores per Evaluator for each of the 14 statements contained in the Evaluation Questionnaire.

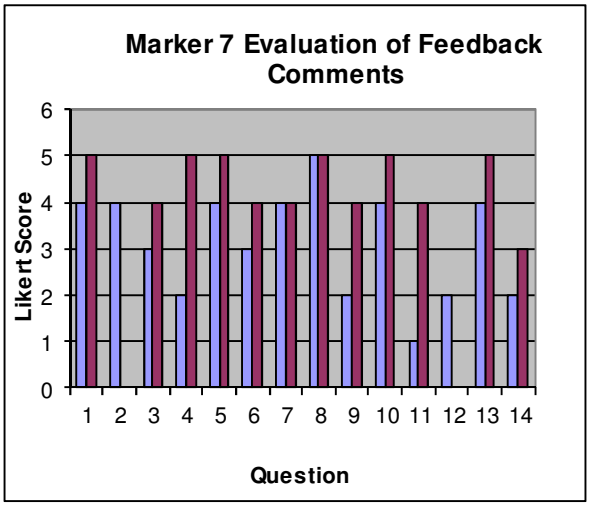

Marker 2 Evaluation of Feedback Comments
Likert Score
0 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Question


Marker 3 Evaluation of Feedback Comments
Likert Score
0 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Question


Marker 4 Evaluation of Feedback Comments
Likert Score
0 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Question


Marker 5 Evaluation of Feedback Comments
Likert Score
0 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Question


Marker 6 Evaluation of Feedback Comments
Likert Score
0 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Question


Marker 7 Evaluation of Feedback Comments
Likert Score
0 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Question

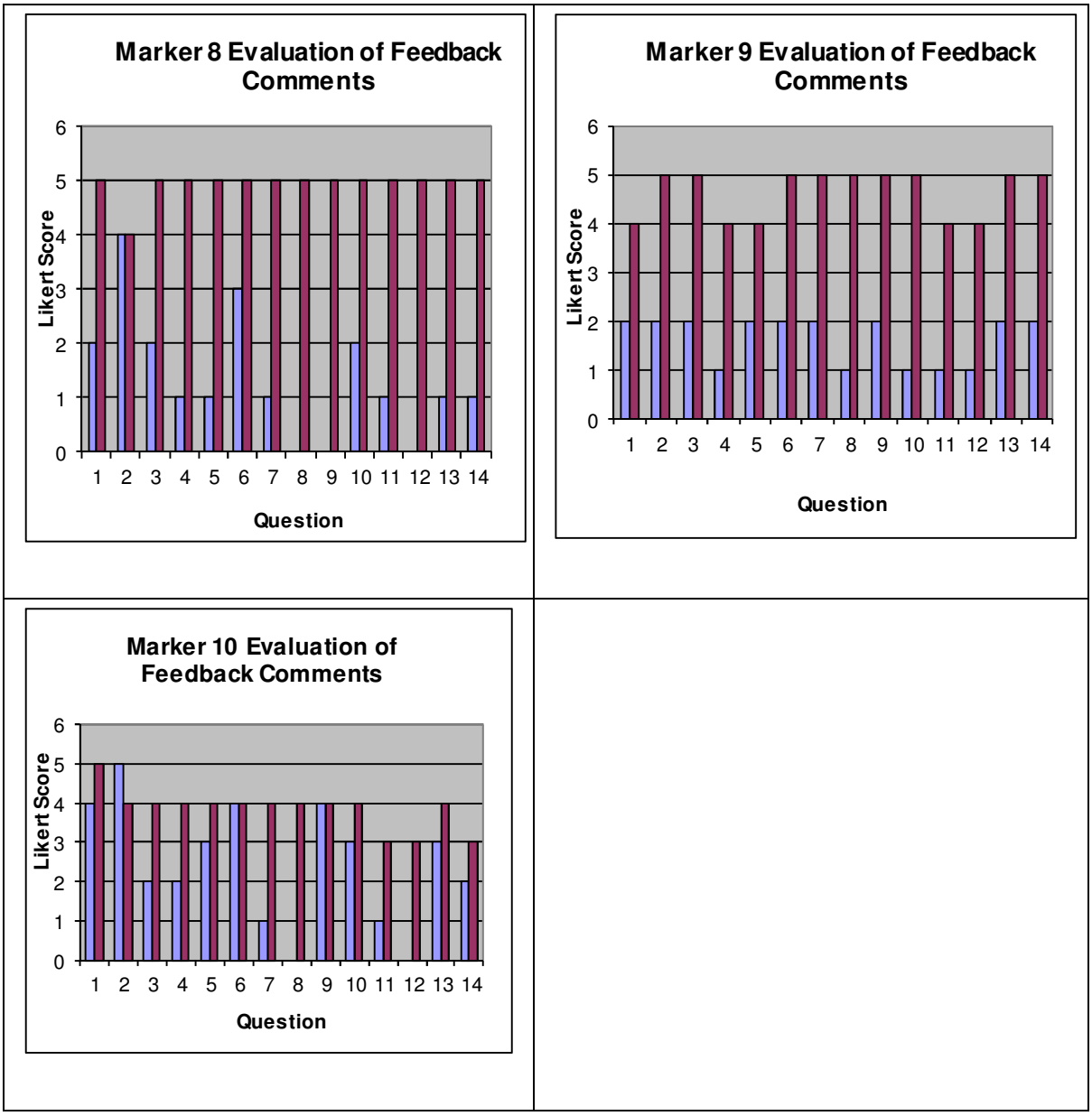


Figure 6.7 Modal Likert scores per evaluator for each of the 14 statements contained in the evaluation questionnaire

The Likert scores from each evaluator were collated for each statement in the questionnaire. The medians for the human and tool generated comments were calculated. The result is illustrated in Figure 6.8 and Figure 6.9 presents a summary of the raw data. An analysis of these figures shows that the comments generated by the tool are rated consistently equal to or higher than those that were

human-generated. This is the case for all of the 13 statements (note statement 2 cannot be considered in this analysis).

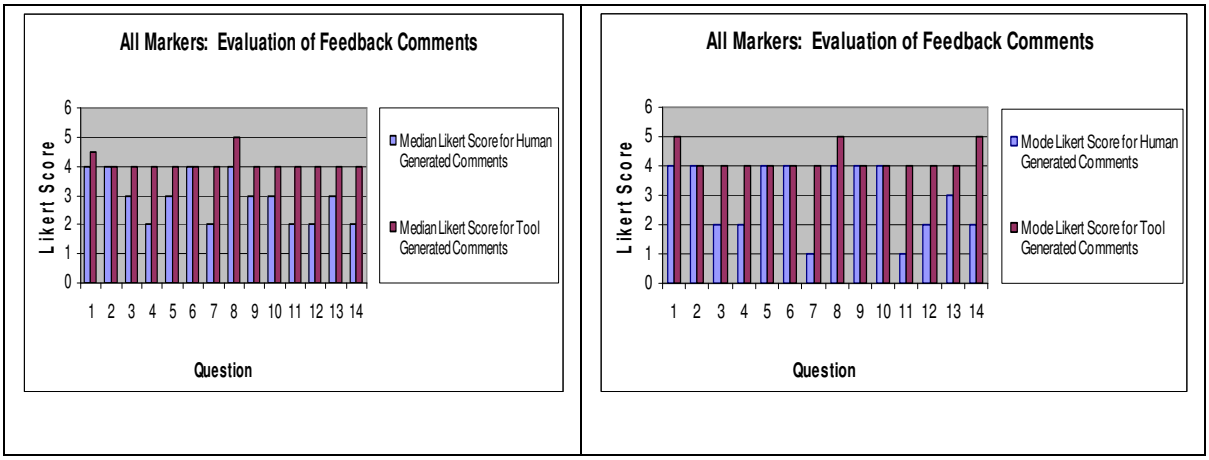


Figure 6.8 Median and mode Likert scores for all evaluators for each of the 14 statements

	N	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
median human	31	4	4	3	2	3	4	2	4	3	3	2	2	3	2
median tool	32	4.5	4	4	4	4	4	4	5	4	4	4	4	4	4

Figure 6.9: A table showing the median Likert Score for both human and tool-generated comments.

Two statistical tests were applied to these results to determine if these differences were statistically significant. These were a sign test and a Mann-Whitney U test. Figures 6.10. and 6.11 tabulate these results respectively. The null hypothesis for the sign test was:

H_0 : The Likert scores for the tool-generated comments are distributed such that half the scores lie above the population median.

The sign test results show that, for all questions with the exception of question11, there is no significant difference in Likert scores for the tool-generated comments. The test shows that for question 11, encapsulation of all pertinent feedback, the Likert scores for the tool-based comments are significantly higher than those that were human-generated. The sign test leads to the conclusion that the tool-generated comments were perceived to be at least as good as those that were human-generated and , for question 11, better.

The null hypothesis for the Mann-Whitney U test was:

H_0 : The distribution of Likert scores is the same across the human-generated comments as it is for the tool-based comments.

The SPSS (IBM 2010) tool for statistical analysis was used to conduct the Mann-Whitney U test. It revealed that there is a statistically significant difference in the distribution of Likert scores across the human-generated comments compared to tool-based comments. The direction of the difference is illustrated in Figure 6.9 and revealed that the Likert scores for the tool generated comments are significantly higher than for those that were human-generated (Appendix G provides further analysis, including a one sided (upper and lower) test). The Mann-Whitney U test produced an observational significance value of p=0.000 for thirteen of the fourteen questions. The test leads to the conclusion that the tool-generated comments were perceived to be better than those that were human-generated.

	Pop size	Pop Median	Number of Likert scores for the tool-generated comments that are greater than the population Median	Sample Size	Proportion of observations above the median Pm	Z	conclusion
Q1	63	4	16	32	0.5	0	Accept H_0
Q2	63	4					
Q3	63	4	14	32	0.4375	-0.70711	Accept H_0
Q4	63	4	12	32	0.375	-1.41421	Accept H_0
Q5	63	4	14	32	0.4375	-0.70711	Accept H_0
Q6	63	4	12	32	0.375	-1.41421	Accept H_0
Q7	63	4	14	32	0.4375	-0.70711	Accept H_0
Q8	63	4	17	32	0.53125	0.353553	Accept H_0
Q9	63	4	14	32	0.4375	-0.70711	Accept H_0
Q10	63	4	13	32	0.40625	-1.06066	Accept H_0
Q11	63	3	26	32	0.8125	3.535534	Reject H_0
Q12	63	4	12	32	0.375	-1.41421	Accept H_0
Q13	63	4	13	32	0.40625	-1.06066	Accept H_0
Q14	63	3	20	32	0.625	1.414214	Accept H_0

Figure 6.10 Sign test results comparing medians for tool-generated comments with those that were human generated.

Test Statistics

	Q1	*Q2*	Q3	Q4	Q5	Q6	Q7
Mann-Whitney U	224.000	*325.500*	183.000	164.000	214.500	243.500	101.500
Asymp. Sig. (2-tailed)	.000	*.013*	.000	.000	.000	.000	.000
Exact Sig. (2-tailed)	.000	*.013*	.000	.000	.000	.000	.000
Exact Sig. (1-tailed)	.000	*.007*	.000	.000	.000	.000	.000

	Q8	Q9	Q10	Q11	Q12	Q13	Q14
Mann-Whitney U	183.000	151.000	199.000	54.000	100.000	100.000	86.500
Asymp. Sig. (2-tailed)	.000	.000	.000	.000	.000	.000	.000
Exact Sig. (2-tailed)	.000	.000	.000	.000	.000	.000	.000
Exact Sig. (1-tailed)	.000	.000	.000	.000	.000	.000	.000

Mann-Whitney U-test : Hypothesis Test Summary		
Null Hypothesis	Significance	Decision
The distribution of Q1 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis
The distribution of Q2 is the same across the categories of human and tool generated comments	*.013*	*Reject the null hypothesis*
The distribution of Q3 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis
The distribution of Q4 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis
The distribution of Q5 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis
The distribution of Q6 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis
The distribution of Q7 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis
The distribution of Q8 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis
The distribution of Q9 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis
The distribution of Q10 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis
The distribution of Q11 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis
The distribution of Q12 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis
The distribution of Q13 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis
The distribution of Q14 is the same across the categories of human and tool generated comments	.000	Reject the null hypothesis

Asymptotic significances are displayed. The significance level is .05.

Alpha is 0.05 and the CI level is 95%.

Figure 6.11 Results utilising the Mann-Whitney U test.

Figures 6.12, 6.13 and 6.14 illustrate the median and modal returns for all evaluators against the criteria of quality, relevance and coverage respectively. The criterion of coverage is the one where there is the most difference between the human- and tool-generated comments. The response to statement 14 “this set of comments would have been sufficient to replace the type of comments that I gave during stage 1 of this evaluation” is interesting as it contradicts a strong view expressed within the student evaluation (a point that will be picked up in the next section where student evaluation is discussed). The evaluators disagreed with this statement when viewing comments that were human-generated but ‘agreed’/’strongly agreed’ when evaluating those that were tool-based. The evaluators’ comments also offer an insight into how the evaluators themselves felt about the feedback comments that they generated after reflecting upon those generated by the tool, their judgement being that the tool-based comments were at least as good as those that they had produced themselves. This is expanded upon in chapter 7.

On the criterion of quality, removing responses for question 2 which related to conciseness, the tool performs favourably when compared to the human markers.

On the criterion of both relevance and coverage the tool performs well in comparison with the human markers. All evaluators rated the tool’s comments as higher or equal to those generated by a human.

Statement	Median Likert Score for Human Generated Comments	Median Likert Score for Tool Generated Comments	Mode Likert Score for Human Generated Comments	Mode Likert Score for Tool Generated Comments
Q1. The comments contained in this set are clear.	Agree (4)	Agree/Strongly Agree (4.5)	Agree (4)	Strongly agree (5)
Q3. The set of comments provide sufficient detail in order for a student to know what concept or issue is being fed back upon.	Neither Agree Nor Disagree (3)	Agree (4)	Disagree (2)	Agree (4)
Q4 The set of comments provide sufficient detail in order for a student to know what further work they need to undertake.	Disagree (2)	Agree (4)	Disagree (2)	Agree (4)
Q5 The Set of comments will help the student with his/her learning	Neither Agree Nor Disagree (3)	Agree (4)	Agree (4)	Agree (4)

Figure 6.12 A breakdown of the median Likert scores for the quality criterion

Statement	Median Likert Score for Human Generated Comments	Median Likert Score for Tool Generated Comments	Mode Likert Score for Human Generated Comments	Mode Likert Score for Tool Generated Comments
Q6 The comments contained in this set are relevant for this type of assignment brief and the associated indicative learning outcomes.	Agree (4)	Agree (4)	Agree (4)	Agree (4)
Q7 The comments contained in this set address important areas of strength found in the student's submission that is considered to be of significance.	Disagree (2)	Agree (4)	Strongly Disagree (1)	Agree (4)
Q8 The comments contained in this set address important areas of weakness found in the student's submission that is considered to be of significance.	Agree (4)	Strongly Agree (5)	Agree (4)	Strongly Agree (5)
Q9 It is clear which concepts the comments in this set are addressing.	Neither Agree Nor Disagree (3)	Agree (4)	Agree (4)	Agree (4)
Q10 The comments in this set will help the student improve his/her solution.	Neither Agree Nor Disagree (3)	Agree (4)	Agree (4)	Agree (4)

Figure 6.13 A breakdown of the median Likert scores for the relevance criterion

Statement	Median Likert Score for Human Generated Comments	Median Likert Score for Tool Generated Comments	Mode Likert Score for Human Generated Comments	Mode Likert Score for Tool Generated Comments
Q11 This set of comments, when viewed in its entirety, fully encapsulates all pertinent feedback needed for the student to recognise where there are areas of strength in the student submission.	Disagree (2)	Agree (4)	Strongly Disagree (1)	Agree (4)
Q12 This set of comments, when viewed in its entirety, fully encapsulates all pertinent feedback needed for the student to recognise where there are areas of weakness in the student submission.	Disagree (2)	Agree (4)	Disagree (2)	Agree (4)
Q13 This set of comments would provide a useful enhancement to the type of comments that I gave during stage 1 of this evaluation.	Neither Agree Nor Disagree (3)	Agree (4)	Neither Agree Nor Disagree (3)	Agree (4)
Q14 This set of comments would have been sufficient to replace the type of comments that I gave during stage 1 of this evaluation.	Disagree (2)	Agree (4)	Disagree (2)	Strongly Agree (5)

Figure 6.14 A breakdown of the median Likert scores for the coverage criterion

6.5 Evaluation by the Student Body

The previous sections presented an evaluation of the tool's formative comments by a team of evaluators. This section presents a student evaluation.

The evaluation took place in a tutorial session with 30 students. They were presented with feedback comments generated by the tool from a sample of the cohort's coursework. The questionnaire was completed via an Electronic Voting System (EVS). This provided an evaluation that was timely whilst preserving student anonymity. A student cannot be identified from the collated electronic response. Figure 6.16 tabulates the results for each of the 11 statements contained in the questionnaire.

Statement 1 is a null statement and was presented to the students as a means of ensuring that the EVS system was working correctly and that the students were able to interact with it.

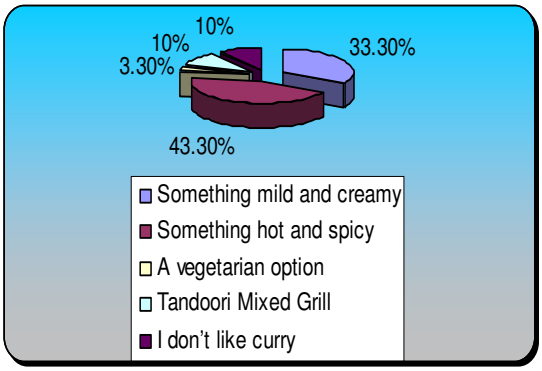
The first notable difference between the students' and evaluators' returns is that the students utilized the full range of Likert ratings more than the evaluators. One explanation for this could be that each undergraduate in the cohort, engaging with the subject for the first time, will be at differing stages in their learning and understanding of the topic area being assessed. Consequently, the same set of feedback comments could have a resonance with some students and less so for others. Conversely, each member of the evaluative team had a significant amount of experience both in the topic area and in teaching/assessment experience. Hence, for this context, it might be reasonable to expect the evaluators' returns to migrate towards a significant consensus.

The students either 'agreed' or 'strongly agreed' with statements stating that the comments were helpful (statement 2, 50%), clear (statement 3, 43%) relevant (statement 4, 47%) and that the feedback was clear on the concepts being

addressed (statement 5, 57%). Whilst 57% of students 'agreed' or 'strongly agreed' that the feedback would help the student to improve the solution (statement 6) they were less convinced that it would lead them to undertake further research into the topic area (statement 7, 38% disagreed or strongly disagreed). One explanation for this could be the use of the term 'research'. The students were year 2 undergraduates and perhaps this term was being contextualised against research contained in the development of a dissertation project as opposed to finding out more about the topic area being assessed. The students indicated that the tool performs better in identifying the strengths (statement 8, 57%) of the submission than the weaknesses (statement 9, 31%). Statements 10 and 11 asked the students to consider the possibility of replacing feedback from a human (tutor) with that generated by the tool. Whilst 62% of the students agreed or strongly agreed that the tool's feedback would enhance that produced by the tutor (statement 10) 66% disagreed or strongly disagreed that the tool's feedback could replace that from a human in its entirety. Despite a positive response to the feedback generated by the tool this points to an underlying distrust in its appropriateness when applied, in isolation, to their personal formative assessment. In contradiction, the team of evaluators indicated that the tools' comments were sufficient to replace those that were human-generated.

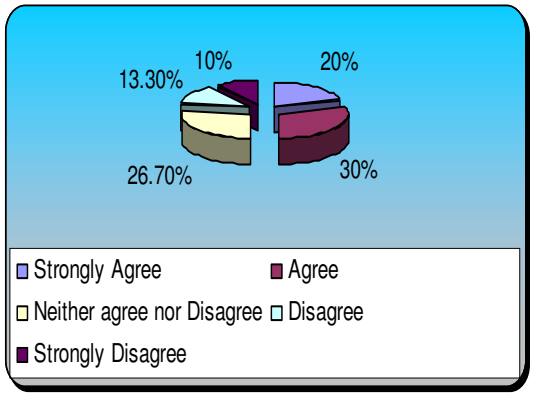
1.) When I go out for a curry I order (multiple choice)

	Responses (percent)	(count)
Something mild and creamy	33.33%	10
Something hot and spicy	43.33%	13
A vegetarian option	3.33%	1
Tandoori Mixed Grill	10%	3
I don't like curry	10%	3
Totals	100%	30



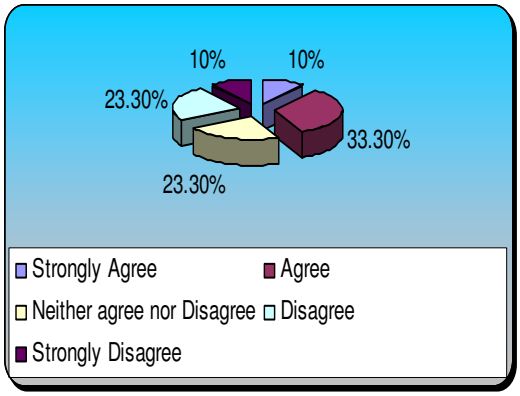
2.) The feedback presented to me is helpful (multiple choice)

	Responses (percent)	(count)
Strongly Agree	20%	6
Agree	30%	9
Neither agree nor Disagree	26.67%	8
Disagree	13.33%	4
Strongly Disagree	10%	3
Totals	100%	30



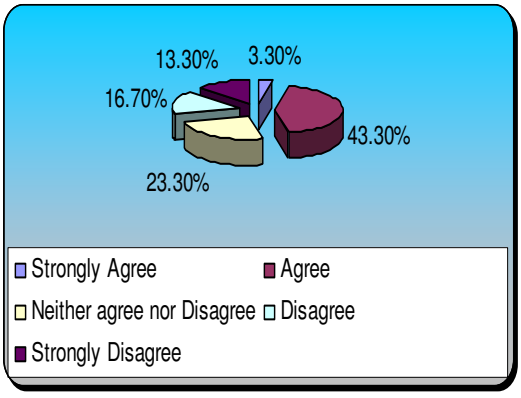
3.) The Feedback Presented to Me is Clear (multiple choice)

	Responses (percent)	(count)
Strongly Agree	10%	3
Agree	33.33%	10
Neither agree nor Disagree	23.33%	7
Disagree	23.33%	7
Strongly Disagree	10%	3
Totals	100%	30



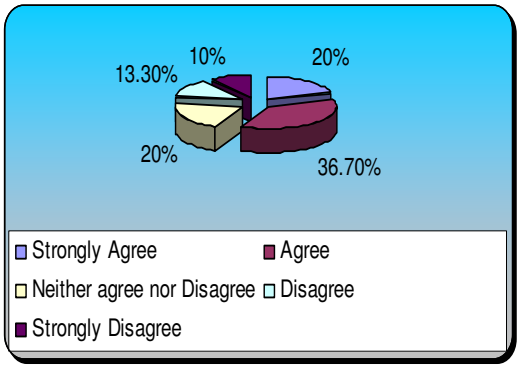
4.) The feedback presented to me is Relevant to my Solution (multiple choice)

	Responses (percent)	(count)
Strongly Agree	3.33%	1
Agree	43.33%	13
Neither agree nor Disagree	23.33%	7
Disagree	16.67%	5
Strongly Disagree	13.33%	4
Totals	100%	30



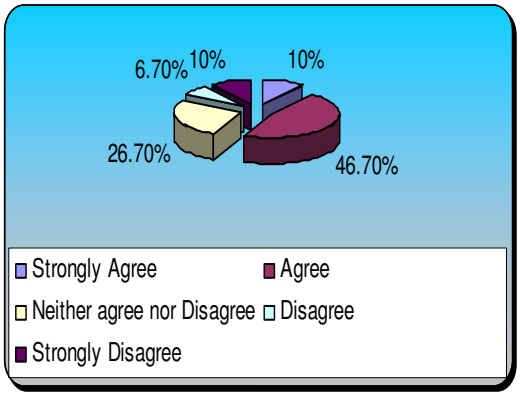
5.) It is clear to me what concepts the feedback is addressing (multiple choice)

	Responses (percent)	(count)
Strongly Agree	20%	6
Agree	36.67%	11
Neither agree nor Disagree	20%	6
Disagree	13.33%	4
Strongly Disagree	10%	3
Totals	100%	30



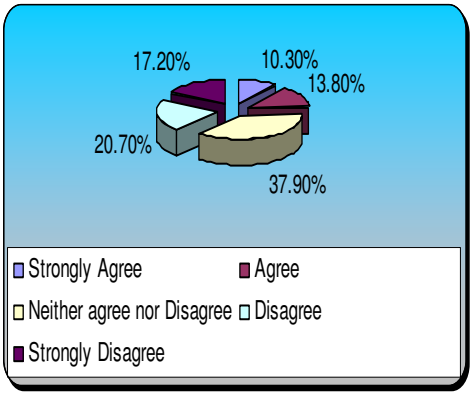
6.) The feedback presented to me will help me to improve my solution (multiple choice)

	Responses (percent)	(count)
Strongly Agree	10%	3
Agree	46.67%	14
Neither agree nor Disagree	26.67%	8
Disagree	6.67%	2
Strongly Disagree	10%	3
Totals	100%	30



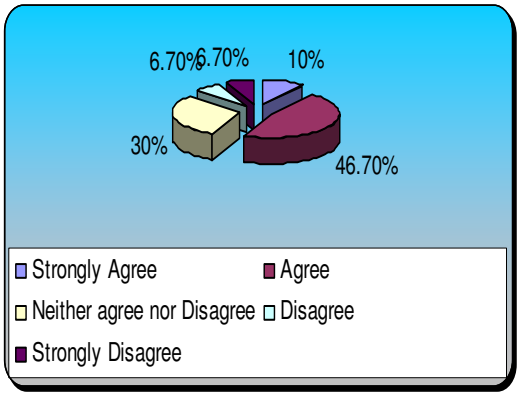
7.) I will use this feedback to research further into this topic area (multiple choice)

	Responses (percent)	(count)
Strongly Agree	10.34%	3
Agree	13.79%	4
Neither agree nor Disagree	37.93%	11
Disagree	20.69%	6
Strongly Disagree	17.24%	5
Totals	100%	29



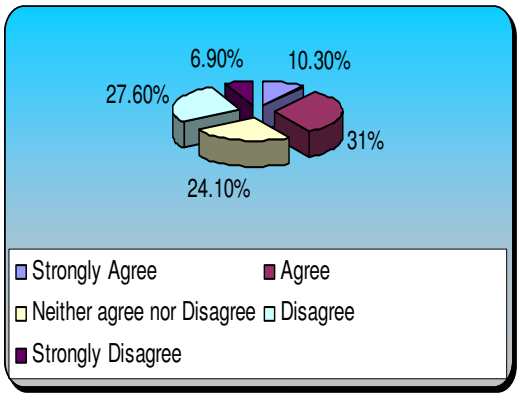
8.) The feedback has helped me identify the Strengths of My Submission (multiple choice)

	Responses (percent)	(count)
Strongly Agree	10%	3
Agree	46.67%	14
Neither agree nor Disagree	30%	9
Disagree	6.67%	2
Strongly Disagree	6.67%	2
Totals	100%	30



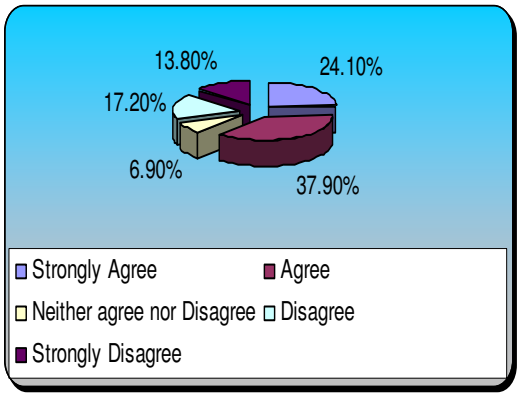
9.) The feedback has helped me identify the weaknesses contained in my submission (multiple choice)

	Responses (percent)	(count)
Strongly Agree	10.34%	3
Agree	31.03%	9
Neither agree nor Disagree	24.14%	7
Disagree	27.59%	8
Strongly Disagree	6.90%	2
Totals	100%	29



10.) The feedback represents a useful enhancement to that which I received from my tutor (multiple choice)

	Responses (percent)	(count)
Strongly Agree	24.14%	7
Agree	37.93%	11
Neither agree nor Disagree	6.90%	2
Disagree	17.24%	5
Strongly Disagree	13.79%	4
Totals	100%	29



11.) The feedback I received is sufficient enough for it to replace that which I received from my tutor (multiple choice)

	Responses (percent)	(count)
Strongly Agree	10.34%	3
Agree	10.34%	3
Neither agree nor Disagree	13.79%	4
Disagree	24.14%	7
Strongly Disagree	41.38%	12
Totals	100%	29

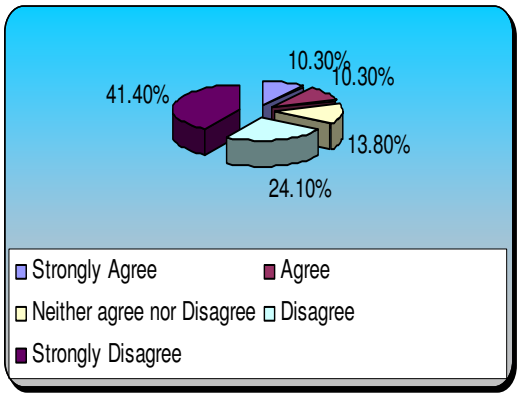


Figure 6.15 Student evaluation of the tool-generated comments

6.6 Summary and Conclusions of the Results.

This chapter described several phases in the evaluation of this research. The first was to collect a sample of human-generated comments and remove those that came from a marker who was seeing the student submission significantly differently to the rest of the team. The second was to conduct an experiment to ensure comparability between the team of evaluators. The third was to conduct an experiment to compare the tool-generated with human-generated comments. The fourth was to conduct a student evaluation of the tool-generated comments.

Summative grades from a common set of student work showed two markers to be grading significantly different to the rest of the team. Consequently their feedback comments were removed from the remainder of the evaluation. A Z test identified the two markers. The AC1 statistic failed to identify them as the variation in percentage grades fell within a single ('A') alpha grade (<=70% mark <=100%).

The AC2 statistic indicated that all members of the team were rating a common set of formative comments consistently for 13 of the 14 questionnaire statements. Hence, these 13 statements were used to compare ratings between the tool and human-generated comments.

The results from the team of evaluators indicate that the comments generated by the tool are at least as good as those generated by the marking team. This is true for the three criteria of quality, relevance and coverage. No conclusion can made regarding the tool's relative performance on the issue of conciseness as there was not a uniform consensus upon this from the evaluative team. The student body rated the tool-generated comments favourably, though they noted that the tool did better at reinforcing strengths in the submission than identifying weaknesses. The students felt that the tool-generated comments would complement that received by their tutor but indicated a lack of confidence in the tool being able to replace the tutor's comments in their entirety. This was not the view held by members of the evaluative team who felt that they could replace the human-generated comments. This issue will be reflected upon later in Chapter 7.

Surveying the students and a team of expert markers ensured that the views of both key stakeholders were included in the evaluation. The logistics of using a team of expert markers was challenging. The two phases of marking the student work followed by evaluating a set of feedback comments required a significant time commitment from the markers. This included the time taken to undertake the

marking and evaluation in addition to maintaining a commitment to the project in between both phases.

In conclusion, the evaluation indicates that the formative comments generated by the tool are at least as good as those generated by a human marker and represent a positive enhancement to the feedback that the students received from their tutor.

Chapter 7. Conclusion and Future Work

7.1 Introduction

The work presented in this dissertation addressed the following research question:

> Given the changing nature of Higher Education, how can we automatically generate high quality feedback for student design task submissions in the form of diagrams.

The question was informed by a review of the literature which identified that:

- the profile of students in Higher Education is changing. Students are better informed, digitally literate and have high expectations regarding their feedback, assessment and support for mobile and remote learning (chapter 1, section 1.1).
- One response to the changing student profile from HEIs is through the use of technology to support learning, teaching and assessment (chapter1, section 1.1).
- Using technology to automate assessment is challenging. The approach taken by existing systems can be characterised by the type of input they accept (free or fixed form), the extent of the automation (fully or semi-automated), and the type of feedback (formative or summative) generated (chapter 2, section 2.3).
- Diagrams are free form items and are difficult to automatically assess (chapter 2, section 2.4.4). The presence of errors and free-form labels exacerbate this difficulty (chapter 2, section 2.4.4.1). Design diagrams and their implementations are examples of coursework submissions from undergraduate computer science students (chapter 1, section 1.1).

- The interface between a design diagram and its implementation is an area of potential inconsistency in the development of computer systems (chapter 2, section 2.4.5).
- The field of model differencing has synergies with automated systems that compare diagrams (chapter 2, section 2.4.4.2).
- Evaluation of existing systems utilise input from tutors, students or both (chapter 2, section 2.5).

The review identified the question of how a student-produced implementation could be used in the automatic assessment of a design diagram to produce formative feedback.

7.2 Contributions

This dissertation has addressed the more general problem of how to automatically generate high quality feedback from paired artefacts with the same referent. It defined such artefacts and their constituent features (chapter 3, section 3.3). It applied the paired artefacts approach to the design/implementation context using an assessment tool applied to a set of student submitted coursework. Each coursework consisted of two artefacts: a design diagram and its accompanying implementation. The tool generates formative feedback based upon the features contained in the artefacts. Features are labelled as being either consistent, superfluous or missing. Feedback positively reinforces consistent features whilst superfluous and missing features are reported as errors.

The dissertation also developed a method for evaluating formative feedback comments. Comments were evaluated by both the students and a team of expert markers. The experts compared human-generated with tool-generated feedback comments produced by the assessment tool while the students evaluated

feedback generated by the tool from an analysis of their submission. The evaluation showed that the feedback from the tool was widely regarded as good, if not better, than that produced by the human markers.

Consequently, the research contained in this dissertation makes the following significant contributions:

- It defines criteria for categorising automated assessment tools.
- It presents a method for automating the assessment of design diagrams by utilising both their implementations and established work that has identified known errors made by novice designers.
- It provides a definition of high quality formative feedback and presents a novel and robust method for its evaluation.
- It presents the generic case by defining terms for multiple artefacts and their assessment.
- It describes an automated assessment tool that generates formative feedback.

7.2.1 Classification of Automated Assessment Tools

The dissertation has identified the core characteristics of tools that automate assessment (chapter 2, section 2.3). This is helpful when considering their adoption as many differ in their approach and the type of feedback generated. A categorisation of such systems was developed using three characteristics: the type of student submission (free or fixed form), the extent of the automation (fully or semi-automated) and the type of feedback generated (formative or summative). Automated assessment tools identified in the literature review were categorised according to these characteristics.

7.2.2 Automated Assessment of Diagrams

The dissertation has identified five challenges for the automated assessment of diagrams: the support for a student to draw and submit a diagram, the support for a tutor to submit a marking scheme, a mechanism to compare a student diagram with a model solution, a mechanism to cope with extraneous/erroneous data and a mechanism to provide feedback to the student (chapter 2, section 2.4.1).

The dissertation has presented a method for automating the formative assessment of student diagrams. The method adopts a blended approach through initially searching for typical errors in the student design before comparing the diagram with its implementation. One benefit of this approach is that it removes the need for a tutor-supplied model answer. Feedback on the comparison offers the student formative support when the development of their solution moves from high to low levels of abstraction. Two potential mechanisms for the comparison have been presented: design-centric and code-centric. The limitations of model differencing, reverse and forward engineering to compare artefacts have been highlighted. This is useful to those who wish to develop the mechanisms further.

7.2.3 Defining and Evaluating Good Quality Feedback

This dissertation has presented an approach to evaluating formative feedback that is both novel and easily transferable to other contexts. It required the development of two Likert-based questionnaires: one completed by a team of evaluators and one by a group of students. The evaluators were members of the computer science academic community. This enabled the perspective of both the suppliers and receivers of feedback to contribute to the evaluation.

Definitions for the quality, relevance and coverage of formative feedback comments were defined (chapter 5, section 5.6.1). From this, fourteen evaluative statements have been derived and formed the questionnaire completed by the evaluators.

Tool-generated feedback was compared with human-generated feedback because there are no metrics for objective measures of feedback quality. A bank of student coursework submissions was collated over several years. The bank was divided into two groups: one used for the development of the tool and one used for its evaluation (chapter 5, section 5.3). Dividing the submissions in this fashion ensured that, during evaluation, the tool had not previously seen the student submissions. It also ensured that the development of the assessment heuristic contained in the tool had not been informed by a student submission that was being used in the tool's evaluation.

A random sample of human and tool-generated comments was sent to a team of evaluators who completed the Likert-based evaluative questionnaire. A comparison between human- and tool- generated comments was conducted which concluded that, on the criteria of quality, relevance and coverage the tool performs well in comparison with the human markers. On the criteria of relevance and coverage all evaluators rated the tool's comments as higher or equal to those generated by an expert human (chapter 6, section 6.4).

The questionnaire used with the students focused upon how the feedback comments helped them with their learning. The tool's feedback was received favourably by the students with most students either agreeing or strongly agreeing that it was helpful, clear, relevant and would help them improve their solution (chapter 6, section 6.5).

7.2.4 Multiple Artefacts: the Generic Case for Diagram Comparison

A novel framework has been developed for the generic case of comparing artefacts. An artefact has been defined as a set of features. Definitions have been provided for consistent and superfluous features (chapter 3, section 3.3). Consistent features have been used for positive reinforcement and superfluous features for where more learning is required. Comparing two artefacts requires

visiting each feature contained in the first artefact and comparing it with each feature in the second artefact. The results of the comparison produce a set of formative feedback comments for each artefact pair. The multiple artefacts approach contributes a new perspective to existing automated diagram assessment systems.

7.2.5 The Development of an Automated Assessment Tool

The efficacy of the multiple artefact framework has been demonstrated through a tool that provides a proof-of-concept implementation (chapter 4). The tool was applied to a set of student-submitted artefacts. It compared two artefacts and identified a set of differences and a set of similarities. When the two artefacts represent a design diagram and its accompanying implementation the differences represent errors in the submission. These errors have either been introduced by the implementation (extraneous) or are those features contained in the design that have not been implemented (omissions). The tool generated formative feedback for these features in addition to positively reinforcing the consistency similarities.

7.3 Reflection upon Comparing Artefacts and Generating Feedback

There are a number of places within the diagram comparison and feedback generation process which, in retrospect, might be improved. This section examines the comparison and feedback generation process and suggests where improvements to the process might be appropriate.

Comparing artefacts first requires describing their consistent and superfluous features (in XML). The XML grammar developed is sufficiently robust to describe the artefacts contained in both the developmental and evaluative data sets. It is also flexible enough to facilitate both a comparison and an analysis of an artefact

in isolation. However, using a CASE tool to automate the artefact's description proved challenging, particularly when one artefact represented a design diagram's implementation. Consequently, the artefacts' description was undertaken manually. This was a laborious process.

Reverse engineering the design diagram's implementation did, however, extract many of its static features, describing them using XML. In retrospect, these descriptions could have been used as a first step followed by manually describing the implementation's dynamic features. This semi-automated approach to describing an artefact's features would have potentially reduced the description time.

Comparing two artefacts requires visiting each feature of one artefact and comparing it with all features of the other. The output is a matching score and a list of feedback comments for each feature pair. Guidance on where further learning is needed is generated for low scoring feature-pairs and positive reinforcement for those with high scores. Mid-scoring pairs represent a partial match. Generating appropriate feedback for mid-scoring pairs is challenging. The lower the score the less likely the pair match. A threshold matching score is set below which the features are considered not to match. This approach generates feedback that has been evaluated positively by both the students and the human evaluators. However, feedback from the students indicated that the tool performed better at reinforcing strengths than identifying weaknesses.

This feedback from the students suggests that more work is required in two areas. The first is the comments embedded within the tool that are generated when errors are detected. Further contextualising the comments as to why a feature-pair was

considered not to match would address the students' concern. However, there is a tension here relating to a generic approach to artefact comparison and generating contextualised and specific feedback. Further exploration of this issue would usefully inform the future development of the approach. For example, by refining the mechanism by which a tutor specifies both the features to be compared and the comments to be generated. A second approach is to undertake a review of the matching process and, in particular, the scoring mechanisms and thresholds at which artefacts are considered not to match. Further tuning of these parameters based upon the results of applying them to a comprehensive data set would provide further insight into the matching algorithm and the conditions under which artefacts are considered not to match.

Blending feedback on specific errors in the diagram with information about how the design diagram compares to its accompanying implementation generated positively reinforcing feedback in addition to identifying where further learning was needed. The student evaluation indicated that the feedback had helped them with their learning whilst the evaluators indicated that it was sufficient to replace the feedback they had provided when marking the submissions. However, whilst the students felt that tool-generated comments would complement those given by a human tutor they were not confident in human-generated comments being entirely replaced by those that were tool-generated.

The students feeling the feedback was not sufficient to replace that of the tutor could suggest a possible mistrust of automated feedback. The tool's tendency to emphasise strengths over errors might contribute to this mistrust or at least cause the students to reflect upon its effectiveness. A follow-up survey with the students would be helpful to investigate further what precisely underpinned this concern.

The pedagogic context of the tool is that of providing formative support for the student as he/she moves from high (design) to low (implementation) levels of abstraction. The approach adopted offers the advantage of not needing a tutor-supplied marking scheme as the feedback generated is via a comparison between the student's diagram and the student's implementation. The feedback focuses upon the consistency between them. This is beneficial as novice students can find moving between levels of abstraction challenging. However, a disadvantage is that the approach does not provide feedback upon whether or not the student's submission is correct and meets the expectation of the assignment brief. The need to triangulate between a marking scheme, the student design and the accompanying implementation has been identified as an area of future work.

7.4 Reflection Upon the Evaluative Method

There are a number of places within the evaluative process which, in retrospect, might be improved. This section examines the evaluative process and suggests where improvements to the process might be made.

The feedback from the automated assessment tool was evaluated by comparing the comments it produces with those produced by expert human markers. The evaluation was based on three criteria: relevance, quality and coverage. For all three criteria, the conclusion was that the tool generated comments were perceived to be better than those that were human generated. However, the result was less pronounced with the coverage criterion than for relevance and quality. Within the coverage criterion, a question about conciseness led to most disagreement within the evaluators, with their responses being spread across the full spectrum of the Likert scale from strongly disagree to strongly agree categories. It would be useful to understand why this apparent discrepancy in the results occurred.

A possible reason for this anomaly might be found in the nature of the comments generated by the tool. Most evaluators agreed or strongly agreed that the tool generated comments were clear, helpful and relevant and there was substantial agreement that the tool generated comments encapsulated all feedback pertinent to both the strengths and weaknesses of the submissions. However, the number of tool-generated comments tended to be greater than those provided by the human markers and some tool-based comments were more verbose than the majority of human generated comments. These factors may have influenced the evaluators in how they interpreted the meaning of conciseness. If the evaluation method were to be repeated, greater care should be taken to ensure that a shared understanding of conciseness was achieved.

The evaluative method took great care to ensure that there was consistency within the team for both marking and evaluation. It was felt important to ensure that team members were marking consistently. To do this, all markers were asked to mark a small, common set of submissions and those who viewed them significantly differently to the rest of the team had their comments removed from the remainder of the evaluation. Was this a sensible approach? Differences of opinion are to be expected and removing some of the data not only reduces the amount of data on which to base conclusions but it might lead to skewed data and a higher agreement between human generated comments and tool generated comments.

Therefore, seeking an alternative method of ensuring marking consistency would be helpful. For example, the markers, having marked the sample set could be brought together to discuss their marking and to identify any differences and come to a shared understanding of how to interpret the marking scheme. However, as

the markers were geographically spread over a wide area and time and resources were limited, meant that this approach was not feasible and the method described above was adopted instead.

Asking the evaluators to rate a common set of feedback comments also identified those questions for which there was little agreement between evaluators. Those questions were removed from the data upon which the evaluation was based. Once again, it would be helpful to identify the reasons for the disagreement and attempt to reduce the extent to which this happened and bringing people together to discuss the questions prior to the main evaluation might reduce the effect of this issue.

Finally, the comments sent to the evaluators deliberately did not distinguish human generated comments from tool based comments in an attempt to avoid bias either towards the human comments or the tool comments. However, the tool based comments were both qualitatively and quantitatively different from the human generated comments (see Table 7.0) and the evaluators may have been able to distinguish between the two classes and, inadvertently or otherwise, introduce bias into their evaluation.

Characteristic	**Human comments**	**Tool Comments**
Number of comments	small	Large
Comprehensive	no	Yes
Order of feedback	random	Consistent
Use of vocabulary	diverse	Limited

Table 7.0 The differences between human and tool generated comments

In retrospect, a re-ordering of the comments generated by the tool coupled with a proactive approach to adopting a wider vocabulary (but delivering the same meaning) might help.

However, if the evaluators did differentiate between tool generated and human generated comments, it is not known what effect this had on the evaluation. A follow-up survey of evaluators could help identify how, if at all, this issue influenced the evaluation.

7.5 Reflection from Academic Participators

Engagement with this research led members of the team of evaluators to reflect upon the type of feedback they themselves gave in the context of their professional practice. Below are quotes from three different members of the team illustrating this point. These quotes were not solicited, they were included in the covering letter which accompanied their completed evaluations.

> "I have found this a very interesting exercise to be involved in and I feel sure it has helped me to improve my own assessment skills. It has definitely clarified the difference between "assessment" comments (how you did it) and "improvement" comments (how you could do it better)."

> "It was interesting to see the tool-generated feedback – I thought it was useful in general."

> "It has been interesting to see the comments generated by your assessment tool. Contrasting these with colleagues' comments really highlights the problems we have as academics in providing good quality feedback to help student learning given the time pressures. "

The first comment illustrates a team member reflecting upon their own approach to formative assessment. The tool's output has helped to clarify the distinction between feedback and feed forward. The third comment refers to time pressures

for producing good quality feedback. The tool offers the potential to reduce this time as, once configured, it can be uniformly applied to the cohort's submission.

7.6 Future Work

Whilst this work has made several significant contributions, there are several ways it can be taken forward.

7.6.1 Support for the Tutor to Enter Feedback Comments

The tool does not create the feedback comments; it generates them by selecting from a predetermined list. The choice of which comments to select is made by the tool when comparing the features of one artefact with those of another. Different comments are chosen according to whether the match is strong, intermediate or weak. Further development is needed on the mechanism by which a tutor specifies both the features to be compared and the comment list related to the strength of their match. This could be through the development of a program that aids the tutor in linking the artefact's features and the feedback to be generated for a range of matching scores. The tutor would run this program once at the start of the assessment as a means of configuring the tool. Alternatively, work could be undertaken in exploring whether natural language techniques could be used to automatically generate feedback comments based upon the features found.

7.6.2 Concise vs. Complete Feedback

There was disagreement between the evaluators upon what constitutes concise feedback (Chapter 6, Section 6.3). A follow-up investigation with the evaluators on the trade-off between completeness and conciseness of the feedback generated would usefully inform the future development of the tool. This could be via a questionnaire, a workshop or through establishing a discussion forum. If there were a mechanism to rank the errors, feedback could be generated for those that were top-ranking.

7.6.3 Identifying Weaknesses in the Student Submission

Student feedback (Chapter 6, Section 6.5) indicated that the tool performs better at reinforcing strengths in the submission than identifying weaknesses. A review of the tool's approach to collating and reporting feedback on weaknesses would usefully inform the future development of the tool. This could be via a follow-up discussion with the students to identify the type and form of feedback they felt would have helped them during their learning.

7.6.4 Syntactically Incorrect Artefacts

Whilst the tool is tolerant of syntax errors, there is an inherent assumption in the tool that both the diagrams and source code are syntactically correct. Further research is needed on how to generate feedback where one or both artefacts are syntactically incorrect. One approach would be to pre-process the artefacts with a domain-specific syntax checking tool. For the design/implementation context this could be through the adoption of a lexical analyser, compiler or CASE tool and the feedback generated would focus upon why the artefact is syntactically incorrect and what needs to be done to correct it.

7.6.5 Triangulating Between Artefacts

The developed assessment tool compared two artefacts – a diagram and its implementation. The pedagogic context is that of providing formative support for the student as the learning moves from high to low-levels of abstraction. The advantage of the approach is that it removes the need for a tutor-supplied mark sheet. However, triangulating between the student submission and a further artefact representing a tutor-supplied mark sheet, for example, would be a useful enhancement to the tool. There are three comparisons that could be made. This would offer the potential of generating additional formative feedback that is focused upon how the artefacts meet the expectation of the tutor as expressed in the mark sheet.

7.6.6 Analysing Free-form Labels

In the design/implementation context, the labels for the features contained in both artefacts are determined by the student. This reduces the complexity of label matching and consequently minimal stemming was adopted by the illustrative tool. However, comparing artefacts produced by different authors, for example, the inclusion of a tutor-supplied marking scheme would require a more sophisticated approach to label matching such as that advocated by Thomas *et al.* (2009).

7.6.7 Tagging Artefacts

The tagging of the student submission is a long and laborious task and relates to the limitation of forward and reverse engineering tools, generally. Further investigation is needed into the adoption of reverse and forward engineering tools to automate the description of an artefact's features. This would involve either the development of a tool that analysed the run-time behaviour of the implementation or a sophisticated tool that statically analysed the source code and extracted from it how and where objects were being dynamically created. Alternatively, it may be fruitful to investigate the feasibility of adopting a hybrid approach where the tools are used to describe an artefact's static features and the dynamic behaviour is described manually. This should make the tagging process less laborious.

7.6.8 Follow-Up Survey with the Evaluators

Feedback comments from two markers were removed from the evaluation as they were viewing the student submission differently (statistically significant) to the remainder of the marking team. There may be merit in revisiting these comments and discussing them with the markers in order to inform future developments of the tool.

7.7 Conclusion

This dissertation has identified an important gap in the literature. No existing systems utilise an accompanying implementation when automatically generating formative feedback for a design diagram. The implementation provides an insight into the student's learning as his/her solution moves from high to low levels of abstraction. It provides a different perspective on the diagram that can usefully inform its assessment. Utilising a diagram's accompanying implementation, therefore, represents a new contribution to the development of systems that automate the e-assessment of diagrams.

The design/implementation context is one example of the generic case where two artefacts represent different ways of expressing a solution to the same problem. The multiple artefact concepts and definitions presented in this dissertation were used to develop an assessment framework. An illustrative assessment tool was implemented and applied to a set of student submissions. The feedback generated by the tool was compared with that generated by a set of human evaluators. The method of evaluation was substantial needing to both test for consistency within the evaluative team and to compare human with tool-generated comments. The evaluation method itself is a novel contribution to the field of e-assessment. Analysis of the evaluators' returns concluded that tool-generated formative feedback comments were rated consistently equal to or higher than those that were human-generated. This was the case for 13 questions distributed across the criteria of quality, relevance and coverage. This suggests that there is merit in the multiple artefact concepts developed in this dissertation. It also suggests that there is merit in the methodology developed for evaluating formative feedback comments. However, there is scope for extension and improvement. Suggestions on where to put future effort have been put forward.

References

[1] Adams, K., 2003. Use of Questionmark Assessment Software to Improve Learning and Teaching. *Proceedings of the Sixth International Conference on Computer Based Learning in Science (CBLIS),* 5-10 July 2003 Cyprus. University of Cyprus, Nicosia, Cyprus, pp. 206-214.

[2] Ali, N., Shukur, Z., and Idris S., 2007a. A Design of an Assessment System for UML Class Diagram. *Proceedings of the Fifth International Conference on Computational Science and Applications*, 26-29 August 2007, Kuala Lumpar, Malaysia. IEEE Computer Society pp. 539-544.

[3] Ali, N., Shukur, Z., and Idris S., 2007b. Assessment System for UML Class Diagram Using Notations Extraction. Computer Science Department, University Malaysia, Malaysia. IJCSNS International Journal of Computer Science and Network Security, VOL. 7 No. 8.

[4] Alphonce, C., and Ventura, P., 2003. QuickUML: A Tool to Support Iterative Design and Code Development. *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications OOPSLA '03*, 26-30 October 2003, Anaheim, California USA. New York: ACM Press pp. 80-81.

[5] Alphonce, C., and Martin, B., 2005. Green: a pedagogically customizable round-tripping UML class diagram Eclipse plug-in. *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange eclipse '05,* 16-17 October 2005*,* San Diego California*, USA.* New York USA: ACM Press pp. 115-119*.*

[6] Anderson, L., and Krathwohl, D., 2001. A Taxonomy for Learning, Teaching and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Longman Publishing, Addison Wesley Longman Inc.

[7] Antoniol, G., Caprile, B., Potrich, A., and Tomella, P., 2000. Design-code Traceability for Object-Oriented Systems. *Annals of Software Engineering,* Vol 9, Issue 1-4, pp.35-58.

[8] Baker, D., and Zuvela, D., 2012. Feedforward Strategies in the First Year Experience of On-line and Distributed Learning Environments. *Assessment and Evaluation in Higher Education*, pp.1-11.

[9] Batmaz, F., and Hinde, C., 2006. A Diagram Drawing Tool for Semi-Automatic Assessment of Conceptual Diagrams. *Proceedings of the 10th International Conference on Computer Assisted Assessment*, 4-5 July 2006, Loughborough. Loughborough: Loughborough University, http://hdl.handle.net/2134/4536.

[10] Batmaz F., and Hinde C., 2007 A Web-Based Semi-Automatic Assessment Tool for Conceptual Database Diagrams. *Proceedings of the 6th International Conference on Web-Based Education,* March 14-16 2007, Chaminox, France. ACTA Press: Anaheim, California USA, Vol 2, pp. 427-432.

[11] Beer D. 2011 – [viewed 02-02-13]. At the Heart of the Higher Education Debate: Expectation Inflation: as Demands Rise, Ability to Meet them Declines [online]. Times Higher Education. Available from http://www.timeshighereducation.co.uk.

[12] Bloom, B., 1956. In *Taxonomy of Educational Objectives: the Classification of Educational Goals*, London, Longman 1956.

[13] Bolloju, N., and Leung, F., 2006. Assisting Novice Analysts in Developing Quality Conceptual Models with UML. *Communications of the ACM* Volume 49 No. 7, July 2006. New York USA: ACM Press, pp108-112.

[14] Bolton, P. 2012 [viewed 02-02-2013]. Tuition Fee Statistics: House of Commons Library, Standard Note SN/SG/917 [online]. Available from http://www.parliamnet.co.uk/briefing-papers/SN00917.pdf.

[15] Borland 2008 Code Gear JBuilder 2008 Professional, Eclipse-based Java IDE, Borland.

[16] Boud, D., and Molloy, E., 2012. Rethinking Models of Feedback for Learning: the challenge of Design. *Assessment and Evaluation in Higher Education*, Volume 0 Issue 0, pp.1-15.

[17] Brown, S., Race, P., and Smith, B., 1996. 500 Tips on Assessment. London: Kogan, ISBN 0749419415.

[18] Bull, J., and Danson, M., 2004. Computer Aided Assessment (CAA). *Learning and Teaching Support Network Generic Centre Assessment Series*. ISBN 1-904190-53-7.

[19] Butcher, K., and Kintsch, W., 2004. Learning with Diagrams: Effects on Inferences and the Integration of Information. *Proceedings of Diagrammatic Representation and Inference, Third International Conference, Diagrams 2004.* 22-24 March 2004. Springer-Verlag, pp. 337-340.

[20] Chawathe, S., Rajaraman, A., Garcia-Molina, H., and Widom, J.. 1996. Change Detection in Hierarchically Structured Information. *Proceedings of the 1996 ACM SIGMOD international conference on Management of Data*, June 4-6 1996, Montreal Quebec Canada. New York USA: ACM Press, pp. 493-504.

[21] Chawathe, S., and Garcia-Molina, H., 1997. Meaningful Change Detection in Structured Data. *Proceedings of the 1997 ACM SIGMOD international conference on Management of Data*, May 13-15 1997, Tucson Arizona USA. New York USA: ACM Press, pp. 26-37.

[22] Chikofsky, E., and Cross, I., 1990. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, Vol 7, Issue 1, pp.13-17.

[23] Cohen, J., 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, Vol 20, pp. 37-46.

[24] Conole, G., and Warburton, B., 2005. A review of Computer–Assisted Assessment. Alt-J, Research in Learning Technology. Vol 13, No. 1, March 2005, pp. 17-31.

[25] Cooper, D., Khoo, B., Von Konsky, B., and Robey, M., 2004. Java Implementation Verification Using Reverse Engineering. *Proceedings of the 27th Australasian conference on Computer Science* - Volume 26 ACSC '04, January 2004, Dunedin New Zealand. Darlinghurst, Australia: Australian Computer Society Inc. pp. 203-211.

[26] Culwin, F., 1998. Web Hosted Assessment – Possibilities and Policy. *ACM SIGCSE Bulletin , Proceedings of the 6th annual conference on the teaching of computing and the 3rd annual conference on Integrating technology into computer science education: Changing the delivery of computer science education (ITiCSE '98),* 18-21 August 1998, Dublin Ireland. New York USA: ACM pp. 55-58.

[27] Cummins, R., A., and Gullone, E., 2000. Why we should not use 5-point Likert scales: The case for subjective quality of life measurement. *Proceedings of the Second International Conference of Quality of Life in Cities*. 2000 Singapore. Singapore: National University of Singapore. pp74-93.

[28] Dafoulas, G., 2005. The role of Feedback in Online Learning Communities. *Proceedings of the Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05).* 5-8 July 2005, Kaohsiung, Taiwan. Washington DC USA: IEEE Computer Society, pp. 827-831.

[29] Daly, C., and Waldron, J., 2004. Assessing the Assessment of Programming Ability. *Proceedings of the 35th SIGCSE technical symposium on Computer science education SIGCSE '04*. 3-7 March 2004, Norfolk Virginia USA. New York USA: ACM Press, pp 210-213.

[30] De Pauw, W., Kimelman, D., and Vlissides, J., 1994. Modelling object-oriented Program Execution. *Proceedings of the 8th European Conference on Object Oriented Programming, ECOOP '94,* 4-8 July 1994, Bologna, Italy. Springer pp. 163-182.

[31] Diamond, I., and Jeffries, J., 2001. *Beginning Statistics: An Introduction for Social Scientists*. London: Sage Publications Ltd., ISBN 0 7619 6061 9.

[32] Dixon, P. N., Bobo, M., and Stevick, R. A., 1984. Response differences and preferences for all category-defined and end-defined Likert formats. *Educational and Psychological Measurement*, 44, pp. 61-66.

[33] Eclipse Foundation 2006 Eclipse**,** open source Integrated Development Environment [online] Available at http://www.eclipse.org [Accessed 2006]

[34] Egyed, A., 2007a. Fixing Inconsistencies in UML Design Models. *Proceedings of the 29th International Conference on Software Engineering (ICSE'07)*, May 20-27 2007, Minneapolis. IEEE Computer Society, pp. 292-301.

[35] Egyed, A., 2007b. UML/Analyzer: A tool for the Instant Consistency Checking of UML Models, *Proceedings of the 29th International Conference on*

Software Engineering ICSE'07, May 20-27 2007, Minneapolis. IEEE Computer Society, pp. 793-796.

[36] Fan S., Tanimoto S. 2007. A framework for automated design assessment in online learning. *Proceedings if the 7th IEEE International Conference on Advanced Learning Technologies* (ICALT 2007), IEEE Computer Society, Niigata, Japan, pp 51-53.

[37] Frigon, N., and Mathews, D., 1997. Practical Guide to Experimental Design. John Wiley and Sons Ltd., ISBN 0-471-13919-X.

[38] Goeb, R., McCollin, C., and Ramalhoto, M., 2007. Ordinal Methodology in the Analysis of Likert Scales. *International Journal of Methodology, Quality and Quantity*, 41, pp. 601-626.

[39] Gwet, K.L., 2010. *Handbook of Inter-Rater Reliability.* 2nd ed. Advanced Analytics, LLC, ISBN 978-0-9708062-2-2.

[40] Haley, D., 2008. *Applying Latent Semantic Analysis to Computer Assisted Assessment in the Computer Science Domain: A Framework, A Tool and an Evaluation*. Thesis (P.hd). The Open University.

[41] Haley, D., Thomas, P., Petre, M., and De Roeck, A., 2008. *Using a New Inter-rater Reliability Statistic*. Technical Report No 2008/15, The Open University, 27th August 2008, ISSN 1744-1986.

[42] Harvey, J., (Ed) 1998. Evaluation Cookbook. *Learning Technology Dissemination Initiative*, Institute for Computer Based Learning, Herriot-Watt University, Edinburgh, ISBN 0 9528731 6 8, URL: http://www.icbl.hw.ac.uk/ltdi.

[43] Hayes, A., 2007. The Development of An Automated Assessment Framework. *Assessment in Wales: Practice that Works* [online]. Available at http://www.heacademy.ac.uk/resources/detail/resource_database/casestudies/welsh_case_studies_index (Accessed December 2007).

[44] Hayes, A., Thomas, P., Smith, N., and Waugh, K., 2007a. A Framework for the Automated Assessment of Consistency Between Code and Design. *Proceedings of Informatics Education Conference II*. 29-30 November 2007, Thessaloniki, Greece. Thessaloniki: South East European Research Centre, pp. 370-378.

[45] Hayes A., Thomas P., Smith N., Waugh K. 2007b An Investigation into the Automated Assessment of the Design-Code Interface. *Proceedings of the 12th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2007)*. University of Dundee, 25-27 June 2007. New York USA: ACM Press pp. 324-324.

[46] Higgins, C.A., and Bligh, B., 2006. Formative Computer Based Assessment in Diagram Based Domains. *Proceedings of the11th annual SIGCSE conference on Innovation and Technology in Computer Science Education (ITiCSE'06)*. 26-28 June 2006, Bologna. New York USA: ACM Press pp. 98-102.

[47] Higgins, C.A., Bligh, B., Symeonidis, P., and Tsintsifas, A., 2009. Authoring diagram-based CBA with CourseMarker. *Computers and Education*, Vol 52 Issue 4, pp. 749-761.

[48] Hoggarth, G., and Lockyer, M.. 1998. An Automated Student Diagram Assessment System. *Proceedings of the 3rd annual SIGCSE conference on Innovation and Technology in Computer Science Education (ITiCSE'98),* 18-21 August 1998, Dublin Ireland. New York USA: ACM Press pp. 122-124.

[49] Holland, S., Griffiths, R., and Woodman, M., 1997. Avoiding Object Misconceptions. In Miller J., E (Ed.), *ACM SIGCSE Bulletin, Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education SIGCSE '97*, Volume 29 Issue 1, March 1997.

[50] Hu, C., 2004. Rethinking of Teaching Objects First. *Education and Information Technologies,* Volume 9 Issue 3, September 2004. Kluwer Academic Publishers pp. 209-218.

[51] Iahad, N., and Dafoulas, G., 2004a. The Role of Feedback in Interactive Learning Systems: A comparative Analysis of Computer-Aided Assessment for Theoretical and Practical Courses. *Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT '04).* 30 August - 1 September 2004*,* Joensuu Finland. Washington DC USA: IEEE Computer Society, pp. 535-539.

[52] Iahad, N., Dafoulas, G., Milankovic-Atkinson, M., and Murphy, A., 2004b. E-Learning in Developing Countries: Suggesting a Methodology for Enabling Computer–Aided Assessment. *Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT '04).* 30 August - 1 September 2004*,* Joensuu Finland. Washington DC USA: IEEE Computer Society, pp. 983-987.

[53] IBM Centre for Software Engineering, Details of ODC, [Online] http://www.research.ibm.com/softeng/ODC/DETODC.HTM (Accessed 14th December 2007)

[54] IBM 2010. SPSS Statitistics version 19.0. Available at Centre for Software Engineering, Details of ODC, [WWW]http://www.research.ibm.com/softeng/ODC/DETODC.HTM (14th December 2007).

[55] IEEE Computer Society and the Association of Computing Machinery The Joint Task Force on Computing Curricula 2001 Computing Curricula 2001 Computer Science. In ACM Journal of Educational Resources in Computing Vol. 1, No.3, December 2001.

[56] Jackson, D., 2000. A semi-automated Approach to On-line Assessment. *Proceedings of the 5th Annual SIGCSE/SIGCUE Conference on Innovation*

and Technology in Computer Science Education. 11-13 July 200, Helsinki, Finland.New York USA: ACM Press pp. 164-167.

[57] Jayal, A., and Shepperd, M., 2009. The Problem of Labels in E-Assessment of Diagrams. *ACM Journal on Educational Resources in Computing*, Vol 6, No. 4, Article 12.

[58] Joint Information Systems Committee (JISC). Effective Practice with e-Assessment, 2007.

[59] Jones C., Ramanau R., Cross S. and Healing G. 2010. Net Generation or Digital Natives: Is there a Distinct New Generation Entering University? *Journal of Computers and Education,* Vol 54, Issues 3, April 2010. Elsevier, pp 722-732.

[60] Jordan, S., 2011. Using Interactive Computer-based Assessment to Support Beginning Distance Learners of Science. *Open Learning: The Journal of Open, Distance and e-Learning*, 26:2, pp 147-164.

[61] Joy, M., Luck, M., 1998. Effective Electronic Marking for On-line Assessment. In Proceedings of the 3rd annual SIGCSE conference on Innovation and Technology in Computer Science Education (ITiCSE'98). 18-21 August 1998, Dublin Ireland. New York USA: ACM Press pp. 134-138.

[62] Joy, M., Muzykantskii, B., Rawles, S., and Evans, M., 2002. An Infrastructure for Web-Based Computer-Assisted Learning. *ACM Journal of Educational Resources*, Vol 2, No. 4, pp. 1-19.

[63] Joy, M., Griffiths, N., and Boyatt, R., 2005. The Boss Online Submission and Assessment System. *Journal on educational Resources in Computing (Jeric),* Volume 5, Issue 3, September 2005, article 2.

[64] Kelly, D., and Shepard, T., 2001. A Case Study in the Use of Defect Classification in Inspections. *Proceedings of the 2001 conference of the*

Centre for Advanced Studies on Collaborative research CASCON '01. 5-7 November 2001, Toronto, Ontario, Canada. IBM Press, page 7.

[65] Kelte, U., Wehren, J., and Niere, 2005. A Generic Difference Algorithm for UML Models. Proceedings of the Software Engineering 2005, Lecture Notes in Informatics (LNI), Essen Germany, March , 2005, GI, Vol.64 pp 105-116.

[66] Landis, J., R., and Koch, G., 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33, 159-174.

[67] Lank, E., Thorley, J., and Chen, S., 2000. An Interactive System for Recognising Hand Drawn UML Diagrams. *Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative Research*. 13-16 November 2000, Mississauga, Ontario, Canada. IBM page 7.

[68] Lass, R., Cera C., Bomberger, N., Char, B., Popyack, J., Hermann, N. and Zoski P., 2003. Tools and Techniques for Large Scale Grading using Web-based Off-The-Shelf Software. *Proceedings of the 8th annual SIGCSE conference on Innovation and Technology in computer Science Education (ITiCSE'03)* 30 June – 1 July 2003, Thessaloniki Greece. New York USA: ACM Press pp. 168-172.

[69] Laurillard D. 2012 Teaching as a Design Space: Building Pedagogical Patterns for Learning and Technology. Routledge.

[70] Lewis, J., 2000. Myths about Object-Orientation and its Pedagogy. *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education SIGCSE '00*. 8-12 March 2000, Austin Texas USA. New York USA: ACM Press pp. 245-249.

[71] Lienhard, A., Ducasse, S., and Girba, T., 2007. Object Flow Analysis – Taking an Object-Centric View on Dynamic Analysis. *Proceedings of the 2007 international conference on Dynamic languages: in conjunction with the*

15th International Smalltalk Joint Conference 2007 (*ICDL '07)*. 21-31 August 2007, Lugano, Switzerland. New York USA: ACM Press pp. 121-140.

[72] Liew, C., 2005. Teaching Software Development Skills Early in the Curriculum Through Software Engineering. *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education ITiCSE '05. 27-29* June 2005, Univerisdada Nova de Lisboa, Monte da Caparica, Portugal. New York USA: ACM Press pp. 133-137.

[73] Likert R. 1932 A Technique for Measuring Attitudes. *Archives of Psychology,* 140, pp. 1-55.

[74] Lilley, M., Barker, T., and Britton, C., 2004. The Development and Evaluation of a Software Prototype for Computer Adaptive Testing. *Computers and Education*, Volume 43, Issue 1-2, pp. 109-123.

[75] Lindland, O., Sindre, G., Brasethvick, T., and Sovberg, A., 1994. Understanding Quality in Conceptual Modelling. *IEEE Software,* Volume 11, Issue 2, pp. 42-49.

[76] Lissitz, R.W., and Green, S.,B., 1975. Effect of the number of scale points on reliability: A Monte Carlo Approach. *Journal of Applied Psychology,* 60, pp. 10-13.

[77] Lister, R., Berglund, A., Clear, T., Bergin, J., Garvin-Doxas., Hanks, B., Hitchner L., Luxton-Reilly A., Sanders, K., Schulte, C., and Whalley, J., 2006. Research Perspectives on the Objects-Early Debate. *Proceedings of the 11th annual SIGCSE conference on Innovation and Technology in Computer Science Education (ITiCSE'06).* 26-28 June 2006, University of Bologna, Italy. New York USA: ACM Press pp 146-165.

[78] Matzko, S., Clarke, P., Gibbs, T., Malloy, B., Power, J., Monahan, R., 2002. Reveal: A Tool to Reverse Engineer Class Diagrams. *Proceedings of the*

Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications CRPIT '02, February 2002, Sydney Australia. Darlinghurst, Australia: Australian Computer Society, Inc., pp. 13 – 21.

[79] McKelvie, S.J., 1978. Graphic rating scale – How many Categories? *British Journal of Psychology,* Volume *(*69), pp. 185-202.

[80] McLaughlin, B., 2001. *Java and XML*. 2nd Ed. United States of America: O'Reilly. ISBN: 0596-00197-5.

[81] Merdes, M., and Dorsch, D., 2006. Experiences with the Development of a Reverse Engineering Tool for UML Sequence Diagrams: A Case Study in Modern Java Development. *Proceedings of the 4th International Symposium on Principles and Practice of Programming in Java (PPPJ 2006)*, 30 August – 1 September 2006, Mannheim, Germany. New York USA: ACM Press, pp. 125 – 134.

[82] National Student Survey 2012. Available at: http://www.thestudentsurvey.com/ (accessed 27th August 2012).

[83] Object Management Group 2007 MOF 2.0/XMi Mapping v2.1.1[online] Available at http://www.omg.org/spec/XMI/2.1.1 (accessed 11th November 2008).

[84] Ohst, D., Welle, M., and Kelte, U., 2003a. Difference Tools for Analysis and Design Documents. *Proceedings of the International Conference on Software Maintenance (ICSM'03).* 22-26 September 2003, Portland Oregon USA. Washington D.C. USA: IEEE Computer Society, pp.13-22.

[85] Ohst, D., Welle, M., and Kelte, U., 2003b. Differences Between Versions of UML Diagrams. *Proceedings of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software*

Engineering. (ESEC/FSE 2003). 1-5 September 2003, Helsinki Finland. New York USA: ACM Press, pp. 227-236.

[86] Pallant, J., 2007. *SPSS Survival Manual*. Open University Press, McGraw Hill, ISBN-10: 0 335 22366 4 (pb).

[87] Parr, C. 2012 [viewed 02-02-2013] At the Heart of the Higher Education Debate: Open University Launched British Mooc Platform to Rival US [online]. Times Higher Education. Available from http://www.timeshighereducation.co.uk/422137.arcticle.

[88] Porter, M.F., 1997. An Algorithm for Suffix Stripping. *Readings in Information Retrieval*, Morgan Kaufmann Publishers Inc., pp. 313-316.

[89] Prados, F., Soler, J., Boada, I., and Poch, J., 2011. An Automatic Correction Tool That Can Learn. *Proceedings of Frontiers in Education Conference (FIE)*, 12-15 October 2011, South Dakota USA. Washington DC USA: IEEE computer Society ppF1D-1-1-F1D-5.

[90] Prensky, M.P., 2001. Digital Natives, Digital Immigrants. *On the Horizon*. NCB University Press, Vol (9), No. 5, October 2001.

[91] Quality Assurance Agency for Higher Education, 2011 – [viewed 27/12/2013]. UK Quality Code for Higher Education Chapter B7:External Examining [online]. Available from http://www.qaa.ac.uk/Publications/InformationandGuidance/Pages/quality-code-B7.aspx

[92] Schmidt, M., and Gloetzner, T., 2008. Constructing Difference Tools for Models Using the SiDiff Framework. *Proceedings of the International Conference on Software Engineering (ICSE '08)*, 10-18 May 2008, Leipzig Germany. New Your, USA: ACM Press, pp. 947-948.

[93] Smith, N., Thomas, P., and Waugh K., 2013. Automatic Grading of free-form diagrams with Label Hypernymy. *Proceedings of Learning and Teachibg in Computing and Engineering (LaTICE, 2013)*. 21-24 March 2013, Macau. IEEE Computer Society , pp136-142.

[94] Smith, N., Thomas, P., and Waugh, K., 2004. Interpreting Imprecise Diagrams. *Proceedings of the Third International Conference in Theory and Applications of Diagrams*. 22-24 March 2004, Cambridge, UK. Springer Lecture Notes in Computer Science, pp. 239-241.

[95] Smith, N., Thomas, P., and Waugh, K., 2010. Diagram Interpretation and e-Learning Systems. In: *A.K., Goel, M., Jamnik, and N.H,. Narayanan eds. Proceedings of the 6th International Conference on Diagrammatic Representation and Inference (Diagrams '10)*. 9-11 August 2010, Portland Oregon USA. *Diagrams 2010*, LNAI,. Berlin Heidelberg: Springer-Verlag, pp.331-333.

[96] Soler, J., Boada I., Prados F., Poch J., and Fabregat R. 2010. A web-based e-learning tool for UML Diagrams. In *Education Engineering (EDUCON),* 2010, IEEE, pp. 973-979.

[97] Sommerville, I., 2007. *Software Engineering.* Edition 8. Addison-Wesley.

[98] Stone, R., Batmaz, F., and Hinde, C., 2009. Drawing and Marking Graph Diagrams. *Italics,* Volume 8 Issue 2, June 2009, pp. 45-52.

[99] Stone, R.G., Batmaz, F., and Rickards, T., 2010. A Multi-Touch ER Diagram Editor to Capture Students' Design Rationale. *Proceedings of the World Congress on Engineering and Computer Science (WCECS 2010)* 20-22 October 2010, San Francisco, USA. International Association of Engineers,pp. 252-256.

[100] Striewe, M., Goedicke, M. 2011. Automated Checks on UML Diagrams. *Proccedings of the 16th annual joint conference on Innovation and Technology in Computer Science Education (ITiCSE 2011)* Darmstadt, Germany, June 27-29, 2011, pp 38-42.

[101] Suraweera, P., and Motrovic, A., 2004. An Intelligent Tutoring System for Entity Relationship Modelling. *International Journal of Artificial Intelligence in Education,* **14** (3-4) pp 375-417.

[102] Suraweera, P., and Motrovic, A., 2002. KERMIT: A Constraint-Based Tutor for Database Modelling. *Proceedings of the 6th International Conference on Intelligent Tutoring (ITS 2002).*2-7 June 2002, Biarritz France and San Sebastian Spain. London: Springer-Verlag, pp. 377-387.

[103] Surridge, P., 2006. The National Student Survey, 2006: Findings. Bristol HEFCE. Available at www.hefce.ac.uk/pubs/rdreports/2006/rd22_06 [accessed 6th February 2012].

[104] Terzis, V., and Economides, A., 2011. The Acceptance and Use of Computer Based Assessment. *Computers & Education*, Volume 56, Issue 4, May 2011, pp. 1032-1044.

[105] Thomas, P., 2004. Drawing Diagrams in an Online Examination. *Proceedings of the 8th CAA Conference.* Loughborough: Loughborough University http://hdl.handle.net/2134/1967.

[106] Thomas, P., Waugh, K., and Smith, N., 2005. Experiments in the Automated Marking of ER-Diagrams. *Proceedings of 10th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2005).* 27-29 June Monte de Caparica Portugal. New York USA: ACM Press pp. 158-162.

[107] Thomas, P., Waugh, K., and Smith, N., 2006. Using Patterns in the Automatic Marking of ER-Diagrams. *Proceedings of the 11th annual SIGCSE conference on Innovation and Technology in Computer Science Education (ITiCSE'06), 26-28* June 2006, Bologna, Italy. New York USA: ACM Press pp. 83-87.

[108] Thomas, P., 2007. Diagram Exerciser: A Tool for Learning Data Modelling. *Proceedings of the 12th annual SIGCSE conference on Innovation and Technology in Computer Science Education (ITiCSE'07),* June 2007. http://mcs.open.ac.uk/Diagrams/Publications/ITiCSE_2007.pdf (accessed 30/01/11).

[109] Thomas, P., Smith, N. and Waugh, K. 2007. Computer Assisted Assessment of Diagrams. *Proceedings of the 12th annual SIGCSE conference on Innovation and Technology in Computer Science Education (ITiCSE'07), 25-27* June 2007, Dundee Scotland. New York USA: ACM Press pp. 68-72.

[110] Thomas, P., Smith, N., and Waugh, K., 2008. Automatically Assessing Graph-based Diagrams. *Learning, Media and Technology*, Vol.33, No.3, pp. 249-267.

[111] Thomas, P., Smith, N., and Waugh, K., 2009. The Role of Labels in the Automated Assessment of Graph-based Diagrams. *Proceedings of 23rd ICDE World Conference on Open and Distance Learning*. 7-10 June 2009, Maastricht, Holland.

[112] Thomas, P., Waugh, K., and Smith, N., 2012 Automatically Assessing Free-form Diagrams in E-assessment Systems. *HEA STEM conference*, 12-13 April 2012, Imperial College, London.

[113] Thomasson, B., Ratcliffe, M., and Thomas, L., 2006. Identifying Novice Difficulties in Object Oriented Design. *Proceedings of the 11th annual SIGCSE conference on Innovation and Technology in Computer Science*

Education (ITiCSE '06). 26-28 June 2006, Bologna, Italy. New York USA: ACM Press pp. 28-32.

[114] Tigris (2006) ArgoUMLv0.22: Open Source Software Engineering Tool [online] Available at http://argouml.tigris.org [Accessed 2006]

[115] Tilley, S., 2000. The Canonical Activities of Reverse Engineering. *Annals of Software Engineering*, Volume 9 Issue 1-4, pp. 249-271.

[116] Treude, C., Berlik, S., Wenzel, S., and Kelte, U., 2007. Difference Computation of Large Models. *Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering.* 3-7 September Dubrovnik Croatia. New York USA: ACM Press pp. 295-304.

[117] Tselonis, C., Sargeant, J., and McGee, M., 2005. Diagram Matching for Human-Computer Collaborative Assessment. *Proceedings of the 9th International Conference on Computer Assisted Assessment*. July 2005. Loughborough: Loughborough University.

[118] Tselonis, C., and Sargeant, J., 2007. Domain-specific formative feedback through domain-independent diagram matching. In: Khandia, F. (ed.). *Proceedings of 11th CAA International Computer Assisted Assessment Conference*. 10-11 July 2007, Loughborough. Loughborough: Lougborough University, pp. 403-420.

[119] Tselonis, C., 2008. *Matching Constructed Answers for E-Assessment*. Ph.D. Thesis. University of Manchester, Faculty of Engineering and Physical Sciences.

[120] Tsintsifas, A., 2002. *A Framework for the Computer Based Assessment of Diagram Based Coursework.* Ph.D. Thesis. University of Nottingham, School of Computer Science and Information Technology. .

[121] Uhrig, S., 2008. Matching Class Diagrams: With Estimated Costs Towards the Exact Solution. *Proceedings of the 2008 international workshop on comparison and versioning of software models (CSVM'08)*. 10-18 May 2008, Leipzig Germany. New York USA: ACM Press pp. 7-12.

[122] Wang, Y., Dewitt, D., and Cai, J., 2003. XDiff: An effective change detection algorithm for XML documents. *Proceedings of the 19th International Conference on Data Engineering*. 5-8 March 2003, Bangalore India. IEEE, pp. 519-530.

[123] Waugh, K., Thomas, P., and Smith, N., 2004. Toward the Automated Assessment of Entity-Relationship Diagrams. *Proceedings of the 2nd LTSN-ICS Teaching, Learning and Assessment in Databases Workshop (TLAD)*. 5th July 2004.Edinburgh, Scotland.

[124] Wenzel, S., 2008. Scalable Visualisation of Model Differences. *Proceedings of the 2008 international workshop on Comparison and versioning of software models (CSVM'08)*, 10-18 May 2008, Leipzig Germany. New York USA: ACM Press, pp. 41-46.

[125] Whitelock. D., and Watt, S., 2007. E-assessment: How can we support tutors with their marking of electronically submitted Assignments? *Ad-Lib, Journal for Continuing Liberal Adult Education.* (32), pp. 7-8.

[126] Williams, J., Kane, D., Sagu, S. and Smith, E., 2008. *Exploring the National Student Survey: Assessment and Feedback Issues*. York, The Higher Education Academy.

[127] Xing, Z., and Stroulia, E., 2005. UMLDiff: An Algorithm for Object-Oriented Design Differencing. *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering (ASE'05)*. 7-11 November 2005, Long Beach California USA. New York USA: ACM Press pp. 54-65.

[128] Yannakoudakis, H., Briscoe, T., and Medlock, B., 2011. A new Dataset and Method for Automatically Grading ESOL texts. *Proceedings of the 49th Annual Meeting of the Association of Computational Linguistics: Human Language Technologies (HLT '11)*. June 19-24 2011 Portland, Oregon USA. Association for Computational Linguistics, pp. 180-189.

[129] Yorke, M., 2003. Formative Assessment in Higher Education: Moves Toward Theory and the Enhancement of Pedagogic Practice. *Higher Education*. 45**:** pp. 477-501.

[130] Zhu, H., and Zhou, M., 2003. Methodology First and Language Second: A Way to Teach Object-Oriented Programming. *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (*OOPSLA '03). 26-30 October 2003, Anaheim California USA. New York USA: ACM Press pp. 140-147.

Appendix A

An EBNF grammar for the XML description of an Artefact's Features

Appendix A

EBNF representation of the tagging grammar adopted

Note on notation

{ } denotes repetition

[ ] denotes optionality

History

22nd May 2009 – Created

29th June 2009 – Modified to include IsContainer and IsContainee in ClassTypeDef and to include definitions for ClassIsContainer and ClassIsContainee

29th June 2009 – Modified to include contained typedef

30th June 2009 – convention specified for the tagging of inheritance, aggregation and dependency relationships

30th June 2009 – added "not defined" to the primitive type specifier

2nd July 2009 – added "char" to the primitive type specifier

6th July 2009 – added "setdynamically" to the cardinality descriptor

10th July 2009 – added tags for "package" and "interface"

10th July 2009 – added optional label feature to the relationship schema

10th July 2009 – added the inclusion of an optional package count and interface descriptor count to StructureDescriptionSchema

13th July 2009 – added "UserDefined" to the PrimitiveType schema

13th July 2009 – added "InterfaceName" to the AdjacentRef schema

13th July 2009 – added "InterfaceID" as an optional component of MethodDef

Convention

Inheritance Relationship – label starts at parent and ends at child irrespective of arrow direction

Aggregation Relationship – label starts at container and ends at containee irrespective of arrow direction

Dependency Relationship – label follows the direction of the arrow ie starts at the tail end ends at the arrow head

TaggedArtefact ::= XMLVersionDescriptor, [Comment], StartGrammarTag,
StructureDescriptionSchema, {[ClassSchema]} , {[RelationshipSchema]},
{[PackageDescriptor]}, {[InterfaceSchema]} , EndGrammarTag;

XMLVersionDescriptor ::= "<?xml version=", ' " ', "1.0", ' " ', "encoding=", ' " ', "UTF-8", ' " ' , "standalone=", ' " ', "yes", ' " ', "?>";

StartGrammarTag ::= "<AML>";

EndGrammarTag ::= "</AML>";

Comment ::= "<comment", {[String]}, "/comment>";

ClassSchema ::= ClassStartTag , ClassTypeDef , [{[AttributeTypeDef]} ,
{[MethodTypeDef]} , {[ChildTypeDef]} , {[ParentTypeDef]},
{[ContaineeTypeDef]}, {[ContainedTypedef]}] , ClassEndTag;

ClassTypeDef ::= "id =" , ' " ' , ClassID , ' " ' , "name =" , ' " ' , ClassName , ' " ', "attributeCount =" , ' " ' ,ClassAttributeCount, ' " ' , "methodCount =" , ' " ' , ClassMethodCount , ' " ' , "IsParent =" , ' " ' , ClassIsParent, ' " ' , "childCount =" , ' " ' , NumberOfChildren, ' " ' , "IsChild =" , ' " ' ,ClassIsChild , ' " ' , "ParentCount =" , ' " ' , NumberOfParents, ' " ' , "IsContainer=" , ' " ' , ClassIsContainer, ' " ', "IsContainee=" , ' " ', ClassIsContainee, ' " ', "AdjacentComponents =" , ' " ' , AdjacentCount, ' " ' , {["AdjacentRef =", AdjacentRef]}, ">" ;

ClassStartTag ::= "<class"

ClassEndTag ::= "</class>"

ClassID ::= Number ;

ClassName ::= String;

ClassAttributeCount ::= Number;

ClassMethodCount ::= Number;

ClassIsParent ::= Boolean;

```
NumberOfChildren ::= Number;
ClassIsChild ::= Boolean;
NumberOfParents ::= Number;
AdjacentCount ::= Number;
ClassIsContainer := Boolean;
ClassIsContainee := Boolean;
AdjacentRef ::= ClassID | InterfaceName ;

AttributeTypeDef ::= ”<attribute ” , “id =” , ‘ ” ’, AttributeID  , ‘ “ ‘,  “name =” , ‘ “ ‘ , AttributeName  ,
                     ‘ “ ‘ , “type =” , ‘ ” ’ , AttributeType , ‘ ” ’  ,“/>” ;
AttributeID ::= “att” , ClassID, “.” , Number  ;
AttributeName ::= String;
AttributeType ::= PrimitiveType| UserDefinedType;
PrimitiveType ::= “int” | “char” | “double” | “real” | “String” | “char” |“NotDefined” | ”UserDefined”;
UserDefinedType ::= String ;
MethodTypeDef ::=   “<method” , “id =” , ‘ ” ’  , MethodID,  ‘ ” ’  , “name =” , ‘ “ ‘ ,MethodName, ‘ “ ‘
                    ,  “/>” ;
MethodID ::= “meth” , ClassID | InterfaceID, “.” , Number ;
MethodName ::= String;

ChildTypeDef ::= “<child”,  “id =” , ‘ ” ’  , ChildID , ‘ ” ’  , “class id =”, ‘ ” ’  , ClassID, ‘ ” ’  ,  “/>”  ;
ChildID ::= “child” , ClassID, “.”  ,Number ;

ParentTypeDef ::=  “<parent”, “id =” , ‘ ” ’  , ParentID , ‘ ” ’  , “class id =”, ‘ ” ’  , ClassID, ‘ ” ’ , “/>” ;
ParentID ::= “parent” , ClassID, “.” , Number  ;

ContaineeTypeDef ::= “<containee”, “id =” , ‘ ” ’  , ContaineeID , ‘ ” ’  , “class id =”, ‘ ” ’  , ClassID,
‘ ” ’  ,  “/>”  ;
ContaineeID ::= “containee” , ClassID, “.”  ,Number ;

ContainerTypeDef ::=  “<container”, “ id =” , ‘ ” ’  , ContainerID , ‘ ” ’  , “class id =”, ‘ ” ’  , ClassID, ‘
” ’ , “/>” ;
ContainerID ::= “container” , ClassID, “.” , Number  ;

RelationshipSchema ::=          RelationshipStartTag , “id =” , ‘ “ ‘  , RelationshipID , ‘ “ ‘  ,
                                “name =” , ‘ “ ‘  ,RelationshipName , ‘ “ ‘  ,“nondangling =” , ‘ “ ‘ ,
                                DanglingDescriptor, ‘ “ ‘ “startclassid =” , ‘ “ ‘  , StartClassDescriptor , ‘ “ ‘
                                ,“startcardinality =”, ‘ “ ‘  ,CardinalityDescriptor , ‘ “ ‘  ,“endclassid =” , ‘ “ ‘
                                , EndClassDescriptor , ‘ “ ‘  , “endcardinality =” ,  ‘ “ ‘
                                ,CardianlityDescriptor ,  ‘ “ ‘  , [“label =” , RelationshipLabel ] ,
                                RelationshipEndTag ;
RelationshipStartTag ::= “<relationship”
RelationshipEndTag ::= “/>”
RelationshipID ::= “rel” , Number;
RelationshipName  ::= “inheritance” | “aggregation” | “association” | “dependency” ;
DanglingDescriptor ::=  “BothEndsConnected” | “OneEndNotConnected” |
“BothEndsNotConnected” ;
StartClassDescriptor ::= “none” | ClassID | PackageID | InterfaceID;
CardinalityDescriptor ::= “none” | “setdynamically” | Number;
EndClassDescriptor ::= “none” | ClassID;
RelationshipLabel ::= String;

StructureDescriptionSchema ::= StructureStartTag, “source  =” , DiagramSource, “class count
=”, ‘ ” ’, Number, ‘ “ ‘, “relationship count =” , ‘ “ ‘ , Number ‘ “ ‘, [ “package descriptor count =”, ‘ ”
’, Number. ‘ “ ‘], [ “interface descriptor count = “,  ‘ “ ‘, Number ‘ “ ‘] StructureEndTag;
StructureStartTag ::= “<StructureDescription”;
StructureEndTag ::= ”/>”;
DiagramSource ::= “student diagram” | “student code” | “tutor model solution”;


PackageDescriptor ::=  “< package id =” , ‘ ” ’  , Package ID , ‘ ” ’  , “name =”, ‘ ” ’  ,
PackageName, ‘ ” ’  ,  “/package>”
PackageID ::= “package ” , Number ;
```

PackageName ::= String;

InterfaceSchema ::= InterfaceStartTag, InterfaceDescriptor, {[MethodTypeDef]},
InterfaceEndTag;
InterfaceStartTag ::= “< interface”
InterfaceDescriptor ::= “id=”, ‘ ” ’ , InterfaceID , ‘ ” ’ , “name =”, ‘ ” ’ , InterfaceName, ‘ ” ’, “>” ;
InterfaceID ::= “interface” , Number ;
InterfaceEndTag ::= “</>”;
InterfaceName ::= String;

Boolean ::= “Yes” | “No” ;
Number ::= {Digit} ;
Digit ::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9” ;
String ::= ‘ ” ’ , {Character} , ‘ ” ‘ ;
Character ::= “a” | “b” | “c” | “d” | “e” | “f” | “g” | “h” | “I” | “j” | “k” | “l” | “m” | “n” | “o” | “p” | “q” | “r” | “s” | “t” | “u” | “v” | “w” | “x” | “y” | “z” | “A” | “B” | “C” | “D” | “E” | “F” | “G” | “H” | “I” | “J” | “K” | “L” | “M” | “N”| “O” | “P” | “Q” | “R” | “S” | “T” | “U” | “V” | “W” | “X” | “Y” | “Z” | Digit ;

Appendix B

User Manual and Summary of Feedback Statements

CompareArtefacts

A tool to generate formative feedback

USER MANUAL

October 2013

Introduction

This document constitutes the user guide for an automated feedback tool. It defines the educational context under which the tool operates and provides guidance upon the inputs and outputs of the tool.

Educational Context

The tool is aimed at providing learning support for undergraduate computing students who are studying an object oriented approach to developing software systems. It is designed to provide students with learning support as they move from the high levels of abstraction needed to design a software system to the lower levels required for its implementation. The tool takes as its input two artefacts:

- a UML design diagram.
- its accompanying Java implementation.

The tool does **not** generate a summative grade - its focus is upon generating formative feedback based upon an analysis of the two artefacts. The formative feedback generated takes two forms.

1. Feedback upon the UML class diagram and whether or not it contains errors
2. Feedback upon how the UML design diagram compares with its java implementation

Students using the tool can submit their artefacts as frequently as they feel is useful – the goal of the tool being to provide them with learning support as they develop their system.

Input Requirements

The name of the tool is CompareArtefacts.jar. The tool requires two files as input:-

- StructureDescriptionfromCode.xml
- Structure DescriptionfromDiagram.xml

The files provide the tool with an xml description of the artefacts upon which the student is seeking formative feedback. They are required to be stored locally on the student's C drive in the following local directory:

- C:\\Users\\CompareArtefacts\\Data

The tool requires the artefacts to be described using a fixed, tool-specific, xml-based grammar. The format for this grammar is appended to the end of this manual. A tool to automate the artefacts' description is under development. Currently, the xml description of the artefacts needs to be produced manually.

Output of the Tool

The formative feedback produced by the tool is output to a text file. The file is called

- FormativeFeedback.txt

This is stored in the directory

- C:\\Users\\CompareArtefacts\\Data

The tool generates four types of feedback:

1. General feedback on the design diagram.

Feedback focuses upon the detection of isolated classes and dangling relationships.

2. General feedback on comparing your design diagram with your implementation.

Feedback focuses upon the number of class and the relationships between them.

3. Detailed feedback on classes drawn in the diagram that match those contained in your program.

Feedback focuses upon the signature of the classes contained in both diagrams.

4. Detailed feedback on classes contained in the diagram that do not match with those contained in the program.

Feedback focuses upon the signature of the classes contained in both artefacts.

5. Detailed feedback on relationships drawn in the diagram that match with those contained in the program.

Feedback focuses upon the type of relationship that connects the classes.

6. Detailed feedback on relationships drawn in the diagram that do not match with those contained in the program.

Feedback focuses upon the type of relationship that connects the classes.

Feedback on known, typical errors made by novice developers.

Typical Error	Test Condition	Feedback
Diagram contains classes that are isolated (i.e. not connected to any other entity in the diagram)	noOfClasses >= 1 && IsolatedClassFound == TRUE	"At least one class in your design diagram is shown not to be related to any others. " "You need to do some further reading on how a programme that consists of message passing objects works."
	noOfClasses >= 1 && IsolatedClassFound == FALSE	"Your diagram does not contain any isolated classes. Well done." "This shows that you understand that a program works through objects being related to each other");
	noOfClasses == 0	“Your design diagram does not contain any classes.” “You need to revisit your understanding of object orientation and data encapsulation."
Diagram contains relationships that do not connect two classes (i.e. dangling at one or both ends)	noOfREls >=1 && DanglingRelationshipFound == TRUE	“You have drawn a relationship that does not connect two classes. " "You need to revisit how you identify and represent relationships between objects."
	noOfRels >=1 && DanglingRelationshipFound == FALSE	“All of the relationships that you have identified have a start class and an end class "This is good as it shows that you have understood that relationships are used to connect the classes contained in your diagram."
	noOfRels == 0	“Your design diagram does not contain any relationships." "You need to revisit your understanding of object orientation and how objects are related to each other."

Holistic Feedback on Matching Features

Feature	Test Val	Feedback
Class	None – this feedback is always generated when two artefacts are compared.	"The number Of Classes in your Design Diagram is <classCountInStudentDiag> and in your implementation you have <classCountInStudentCode> (<totalClassCountInBothDiagrams > in total)"
	noOfClassMatches > 1	"There are <noOfClassMatches> classes that match well when comparing your design with your implementation (<noOfClassMatches> from <totalClassCountInBothDiagrams> "
	noOfClassMatches = 1	"There is 1 class that matches well when comparing your design with your implementation"
	noOfNonMatchingClasses > 1	"There are <noOfNonMatchingClasses> (from <totalClassCountInBothDiagrams>) for which a match could not be found "
	noOfNonMatchingClasses = 1	"There is 1 class for which a match could not be found "
Relationship	None – this feedback is always generated when two artefacts are compared.	"The number Of Relationships in your Design Diagram is <relationshipCountInStudentDiag> and in your implementation you have <relationshipCountInStudentCode> (<totalRelCountInBothDiagrams> in total)."
	noOfRelMatches >1	"There are <noOfRelMatches> relationships that match well when comparing your design diagram with your implementation (<noOfRelMatches> from <totalRelCountInBothDiagrams>)."
	noOfRelMatches = 1	"There is 1 relationship that matches well when comparing your design with your implementation (<noOfRelMatches> from <totalRelCountInBothDiagrams>)."
	noOfNonMatchingRelationships > 1	"There are <noOfNonMatchingRelationships> (from <totalRelCountInBothDiagrams>) for which a match could not be found. "
	noOfNonMatchingRelationships = 1	"There is 1 relationship (from <totalRelCountInBothDiagrams>) for which a match could not be found. "

Detailed Feedback - Class Signature : Name

Feature	Signature	Test Val	Feedback	Score
Class	ClassName	Class1Name == Class2Name	“The names of these two classes match well.”	ClassNameScore = 10
		Class1Name != Class2Name	“A significant difference has been detected in the names of these classes.”	ClassNameScore = 0

Detailed Feedback - Class Signature : Attribute

Feature	Signature	Test Val	Feedback	Score
Class	ClassAttribute	noOfAttributesInClass1 == noOfAttributesInClass2	"Both classes contain the same number of attributes."	attributeCountScore = 10
		diff (noOfAttributesInClass1, noOfAttributesInClass2) <=3	“These two classes differ in the number of attributes that each contains."	attributeCountScore = 5
		diff (noOfAttributesInClass1, noOfAttributesInClass2) >4	“There is a significant difference in the number of attributes that these classes contain.”	attributeCountScore = 0
		attributeCountofClass1 == 0 || attributeCountofClass1 == 0	One of these classes contain no attributes: You probably need to revisit your notes on analysis and design and look again at how you allocate data components to a class"	attributeCountScore = 0
		attributeCountofClass1 == 0 && attributeCountofClass1 == 0	"These two classes do not have any attributes: You probably need to revisit your notes on analysis and design and look again at how you identify the data components of a class."	attributeCountScore = 0
		(both classes have the same number of attributes and each class contains identical attribute names) attributeCountOfClass1 == attributeCountOfClass2 && numberOfSimilarAttributes == attributeCountOfClass1	“The attributes in these two classes match well on both name and number.”	attributeNameScore = 10

Feature	Signature	Test Val	Feedback	Score
		(both classes contains within a tolerance (set at 2) the same number of identical attribute names) numberOfSimilarAttributes >= attributeCountOfClass1 - attributeNameTolerance && numberOfSimilarAttributes >= attributeCountOfClass2 - attributeNameTolerance	“There is a good match in the attributes of these two classes with only minor differences between the two.”	attributeNameScore =7
		(all methods in one class have matched with those of another but there are a different number of methods in each class) numberOfSimilarAttributes == smallestOf (attributeCountOfClass1, attributeCountOfClass2)	"Some attributes match well in these two classes but a significant number don't. You probably need to revisit your notes on analysis and design and look again at how you allocate data components to a class."	attributeNameScore =5
		None of the above test conditions	“The attributes contained in these two classes are significantly different.”	attributeNameScore =0

Detailed Feedback - Class Signature : Method

Feature	Signature	Test Val	Feedback	Score
Class	ClassMethod	both classes have the same number of methods and each class contains identical method names methodCountOfClass1 == methodCountOfClass2 && numberOfSimilarMethods == methodCountOfClass1	"There is a good match in both the method name and number for these two classes."	methodNameScore = 10
		both classes contains within a tolerance (value of 2) the same number of identical method names numberOfSimilarMethods >= methodCountOfClass1 - methodNameTolerance && numberOfSimilarMethods >= methodCountOfClass2 – methodNameTolerance	"These two classes match well in their methods both on name and number with only minor differences between the two."	methodNameScore = 7
		all methods in one class have matched with those of another but there are a different number of methods in each class	“Some of the methods match well in these two classes but a significant number don't. You probably need to revisit your notes on analysis and design and look again at how you identify the methods of a class."	methodNameScore = 5

Feature	Signature	Test Val	Feedback	Score
		numberOfSimilarMethods == smallestOf (methodCountOfClass1, methodCountOfClass2)		
Class	ClassMethod	None of the above tests have been satisfied	"The methods described in these two classes suggests that you think these are very different entities. You need to revisit your notes on identifying and implementing objects. "	methodNameScore = 0
Class	ClassMethod	methodCountClass1 == methodCountClass2	"These two classes have the same number of methods."	methodCountScore =10
		Diff(methodCountClass1, methodCountClass2) <= 3	"These two classes differ slightly in the number of methods that each contains."	methodCountScore =5
		Diff(methodCountClass1, methodCountClass2) > 3	"There is a significant difference in the number of methods specified for each class."	methodCountScore = 0
		methodCountClass1 == 0 ||methodCountClass2 == 0	“One of your classes does not contain any methods. This suggests that you probably need to revisit your notes on how you identify the methods of a class. "	methodCountScore = 0
		methodCountClass1 == 0 && methodCountClass2 == 0	“Neither of these two classes contain any methods. This suggests that you probably need to revisit your notes on how you identify the methods of a class.”	methodCountScore = 0

Matching Score For Classes

overallScore = (classNameScore + (methodScore +attributeScore)/2)/2

attributeScore = (attributeCountScore + attributeNameScore)/2

methodScore = (methodCountScore + methodNameScore)/2

Detailed Feedback – Relationship Signature : Type of Relationship

Feature	Signature	Test Val	Feedback	Score
Relationship	Type	Rel1Name == Rel2Name	No feedback comment – test contributes to the score	scoreOnRelationshipType = 10
		Rel1Name != Rel Rel2Name	No feedback comment – test contributes to the score	scoreOnRelationshipType = 0

Detailed Feedback – Relationship Signature : Connecting Classes

Feature	Signature	Test Val	Feedback	Score
Relationship	Connecting Classes	(the two relationships connect the same classes) class1StartName.equals(class2StartName) && class1EndName.equals(class2EndName)	"Your design and program both relating class <class1StartName> and class <class1EndName> with a <rel1Name> relationship."	scoreOnConnectedClasses = 10
		class1StartName.equals(class2StartName) && (class1EndName.equals(class2EndName)==false)	"You need to think about how you have identified the <rel1Name> relationship as in your program <class1StartName> is related to <class1EndName> whilst in your design it is related to <class2EndName>."	scoreOnConnectedClasses = 5
		class1StartName.equals(class2StartName)) == false && class1EndName.equals(class2EndName)	"You need to think about how you have identified the <rel1Name> relationship as in your program <class1StartName> is related to <class1EndName> whilst in your design it is connected to <class2EndName> ."	scoreOnConnectedClasses = 5
		(this case relates to reverse direction of arrows)	“You need to think about how	scoreOnConnectedClasses

Feature	Signature	Test Val	Feedback	Score
		class1StartName.equals(class2EndName)&& class1EndName.equals(class2StartName)	you represent the <rel1Name> as you have changed the meaning of the relationship between class <class1StartName> and class <class1EndName> in your program compared to that contained in your design.”	= 6
		(both relationships connect at least one common class but polarity is reversed) class1StartName.equals(class2EndName)|| class1EndName.equals(class2StartName)	"You have a partial implementation of the relationships between <class1StartName> , <class1EndName> and <class2StartName> , <class2EndName> .”	scoreOnConnectedClasses = 3
		None of the above satisfied.	These relationships are not related.	scoreOnConnectedClasses = 0

Detailed Feedback – Relationship Signature : Cardinality

Feature	Signature	Test Val	Feedback	Score
Relationship	Cardinality	(startCardRel1 == startCardRel2 && endCardRel1== endCardRel2) = TRUE	"The cardinalities of the " +rel1Name +" match well in both your design and your programme."	scoreOnCardinality = 10
		(startCardRel1 == startCardRel2 && endCardRel1== endCardRel2) = FALSE	“You need to think about cardinalities and what they mean as they have changed from what you state in to your design and what you actually implemented in your program.”	scoreOnCardinality = 0

Matching Score for Relationships

overallScore = (scoreOnRelationshipType + (scoreOnConnectedClasses + scoreOnCardinality)/2)/2

Feedback On Matching/Non-Matching Feature Pairs

Feature	Test Val	Feedback
Class	matchFoundforClass1andClass2 == TRUE	“Class <nameClass1> from your program is a close match to Class <nameClass2> from your design."
	matchFoundforClass1andClass2 == FALSE	“Your implementation contains a class called <nameClass1> which is sufficiently different from all those contained in your design diagram to suggest that there is a mis-match between what you have designed and what you have implemented."
Relationship	numOfRelsInDIag1 >0 && numOfRelsInDiag2 >0 && matchFoundforRel1AndRel2 == TRUE	"You have shown that you understand how to implement the relationships that you have identified in your design. Well done" “You have shown this through :- "
	numOfRelsInDIag1 >0 && numOfRelsInDiag2 >0 && matchFoundforRel1andRel2 == FALSE	“The <rel1Name> relationship in your program that connects class <class1Name> with class <class2Name> could not be matched with any relationship in your design. You need to think about how your design matches your implementation for all classes and objects contained in your system."
	numOfRelsInDiag == 0 && numOfRelsInImplementation == 0	“Cannot compare the relationships in your submission as both your design and your implementation do not contain any.”
	numOfRelsInImplementation ==0 && numOfRelsInDiag >0	“Cannot compare the relationships in your design diagram and your implementation as your implementation does not contain any.”
	numOfRelsInDiag == 0 && numOfRelsInImplementation >0	“Cannot compare the relationships in your design diagram and your implementation as your design does not contain any."

Appendix C

Advice Given and the Questionnaire used with the Team of Expert Markers and the Team of Evaluators

Appendix C part 1 Covering Letter to the team of expert Markers

Alan Hayes
Director of Teaching
Department of Computer Science
University of Bath
BA2 7AY

16th November 2010

Dear

Thank you for agreeing to help with this research – it is greatly appreciated. The broad theme of this research is in the area of automated assessment. I have developed a tool that analyses a student submission and provides formative feedback to the student as a consequence of this analysis. The submission consists of a design diagram (UML) and a source code implementation (java).I now need to evaluate the effectiveness of the comments generated by this tool and it is this stage that I am asking for your help. I want to compare and evaluate the comments generated by my tool with those generated by a set of academic colleagues.

The evaluation will take place in two phases. Phase 1 involves the collection of typical expert marker comments which will be used for developing the tool and is not explicitly related to its evaluation. It will involve you looking at a number of (anonymised) student submissions and ask you to provide the written formative feedback that you would ideally have given to the students to help them with their learning. Phase 2 involves evaluating the formative feedback comments generated by my assessment tool. More details on the first phase are provided below in addition to an indicative timescale. Details on the second phase will follow nearer the time.

I hope all is clear but if not, please get back to me.

Many thanks once again for your support.

Regards

Alan

Indicative Timescales

Activity	Completion Date
Phase 1	
Guidance and student assignments sent to colleagues	**November 17th 2010**
Formative Comments returned to Alan	**December 23rd 2010**
Phase 2	
Comments sent to colleagues for evaluation	**January 28th 2011**
Evaluations returned to Alan	**February 18th 2011**

Phase 1 – Providing Formative Feedback Comments

Please find attached the following:-

1) A set of assignment briefs
2) A set of marking schemes
3) 10 assignment submissions where each submission consists of a student design diagram and its accompanying source code implementation. Note that the student assignments have been allocated to you on a random basis. Hence the numbering of the student submissions are not necessarily in a consecutive order.
4) 10 forms for recording your evaluative comments.

The students have submitted their assignment as a component of an introductory undergraduate unit/module in software development. They are asked to produce a UML diagram based upon their analysis of a given scenario. They are also required to implement their design. For many students it will be their first experience of developing systems using object oriented methods. Consequently, they will be making the typical mistakes of novice developers. It is important that the students are not only supported in developing a strong understanding of object oriented concepts but that they also understand the software development process and in particular the link between a design and its implementation.

For each submission in your pack please supply the feedback comments you would provide to a student in order to reinforce/support their learning. Please provide as many comments as you normally would do given the novice nature of the students' backgrounds. If possible please restrict each individual comment to one idea or concept – probably of no more than a single sentence.

Please record your comments on the attached form. Note the form will accommodate 6 comments – please do not treat this as an upper or lower limit on the number of comments you can provide. You should provide the number of comments that you would do normally. If you would normally provide more than 6 comments please add these to the end of the form.

Please note that the form also contains an entry to record a percentage grade. In marking the student submission please can you utilise the marking scheme to determine an overall mark. This will not be used in the formal evaluation of the tool but will provide a useful context on the comments that you generate. For example, feedback comments on a piece of work with a low percentage grade will be very different to those with high percentage grades.

Checklist

1) You have been sent 10 pieces of student work. Please mark all 10 (ideally) or at least 5 (minimum).
2) Please record your marks and comments on a separate form for each piece of student work.
3) One piece of work consists of a design diagram (UML) and an accompanying implementation (java). Please look at both components when marking.
4) You have been sent an assignment brief and a marking scheme. Please can you refer to these when marking the submission
5) Please provide those feedback comments that you would normally provide to the student on the sheet provided. The focus is upon formative feedback – ie those comments that you feel will help the students in their learning.
6) Please note the form will accommodate 6 comments but this is neither an upper nor a lower bound – please use an extra sheet if you need to.
7) Please can you also provide an overall summative grade for the student work in the form of a percentage mark.
8) Please can you return the completed mark sheet to me either by hard copy:-

Alan Hayes
Director of Teaching,
Department of Computer Science,
University of Bath,
Bath BA2 7AY.

Or electronically to

a.hayes@bath.ac.uk

Appendix C part 2 Form for Recording Marks and Comments from the team of expert markers

Assignment Ref	
Marker	
Summative Mark	(please refer to marking scheme for criteria) Comparing Diagram with Model Solution /50 Comparing Diagram with Source Code /50 Total /100
Comment 1	
Comment 2	
Comment 3	
Comment 4	
Comment 5	
Comment 6	

Appendix C – part 3 An example Assignment Brief

Software Development 2 (G106190)

Assignment Title: Practical Task

Submission: On or before Friday 12th January 2007

Report to be submitted to the school office as per school policy.

The completed application should be demonstrated to the lecturer prior to the due date as well as being submitted on CD with the report.

Note that the submission date is Friday of revision week for semester one exams. It is your responsibility to properly manage your time so that completion of the assignment doesn't have any influence on your revision.

Learning Outcomes Tested:

- Demonstrate a good understanding of object concepts such as encapsulation, abstraction, inheritance and polymorphism
- Discuss the properties of software object systems
- Create class definitions that model real world systems
- Create robust software which employs object concepts and techniques
- Use an object oriented programming language to achieve a stated task.

The Scenario

A university employs three different kinds of employee: - lecturers, administrators and researchers. The University is looking to automate its accounts department so that employee's details can be stored and manipulated more efficiently. The new system must be able to store the name, address, telephone number and employee number of each person employed. In addition, the system must also provide a facility that calculates the monthly payment due to each person. All employees are paid on a monthly basis but the method of calculation differs from category to category. Researchers are paid a basic annual salary of £10,000 per year with no additional bonuses and no overtime payments. Administrators are paid a basic annual salary of £15,000 per year and from time-to-time are expected to work overtime for which they are paid £10 for each extra hour worked. Lecturers are paid a basic annual salary of £20,000. They are not expected to work overtime but do receive two additional types of payment: - consultancy and performance related pay. Each hour worked as consultancy for the University is paid at a rate of £20 per hour. Performance related pay is a fixed amount of money awarded to a lecturer each year. This amount is divided into twelve equal instalments and the lecture receives one instalment per month in his/her pay packet.

Caveat
All salaries and methods of payment outlined in the above scenario are entirely fictitious. The author, at the time of writing this assignment, had no prior knowledge of Newport University's pay structure for all grades of employees. Any resemblance to the actual pay scheme used by Newport is entirely coincidental.

Scope of Your Assignment
You are required to implement and report upon a solution to the above scenario. Your solution should use object oriented techniques wherever appropriate. You should perform an OO analysis/design using the UML methodology. The scope of your analysis should incorporate the identification of all objects in the system including their attributes and methods. For each object identified you should provide an appropriate object interface diagram. You should also graphically represent any relationships between the objects that you have identified.

Having completed your analysis and design you are required to provide an implementation written in the Java programming language. Your program should contain the class specification for each of the objects that you identified. For this assignment you can assume that the university employs 20 lecturers, 10 researchers and 10 administrators. You need not concern yourself with storing your data to disk. The main focus of your implementation should concentrate upon manipulating a list (or array/vector) of university employees. This should include the calculation and reporting of the salary to be paid to each employee for this particular month. Your calculation of salary should be based upon the concept of polymorphism.

Deliverables
There will be three deliverables for this project. Two of these deliverables are required to be submitted electronically and one in hard copy/report format. The two electronic submissions are:-

1. The design of your system in **UML** created using the **community edition** of **Poseidon**
2. The java source code

The report is a non-electronic submission and should be handed in to the student office and receipted in the normal way. Marks are distributed as indicated by the mark sheet below. You should note that the two electronic submissions will be used to check for consistency between your design and its implementation and that marks have been allocated for this. Electronic submission should be made on CD which accompanies the report for archiving and moderation purposes.

The Report
Your report should detail the work you have done in order to produce your solution to the scenario. Your report should contain the following sections:-

Introduction
The specification of this assignment has been made deliberately vague and ambiguous. Your introduction should set the scope of the report and state the assumptions that you have during your implementation.

Analysis and Design

This section should document the analysis and design phase of your implementation.

Implementation

This section should contain a print-out of your well-comment, highly modularised and structured source code.

Testing

This section should provide details of the testing that you have performed in order to validate the integrity of your system. It should include testing on an object-by-object basis as well as the final integrated system.

Critical Appraisal

Object technology claims to improve maintenance and re-use of software systems. You should discuss the appropriateness of this claim citing examples, where appropriate, taken from your own implementation.

Group work

There is no scope for group work within this assignment. All work must be carried out on an individual basis.

Hints and suggestions:

- Check with the tutor if you have any queries.
- Do not neglect the report in a coding effort or vice versa.
- Time spent thinking about the problem is NOT wasted.
- Check learning outcomes and grading criteria before during and on completion of the assignment. Do not submit until you are sure you have met them.

Plagiarism and unfair practice

It is dishonest not to acknowledge the work of other people and you open yourself up to the accusation of plagiarism. The text of this assignment **must be in your own words** (not even a sentence or phrase should be taken from another source unless this source is referenced and the phrase placed in quotes).

For more information in respect of plagiarism please refer to the University Assessment Regulations at the following web address:

http://quality.newport.ac.uk

The tutor may decide to submit your assignment to automate plagiarism checks.

Grading Criteria

The overall Mark and Grade for the Assignment will be awarded as follows:

% mark	Grade	Criteria
70%<=mark<=100%	A	A working application is demonstrated which clearly shows that good programming practice has been followed. The report documents a comprehensive design process with consideration of usability, reuse and maintenance and the requirements of the assignment as listed above. This design corresponds with the demonstrated application. The report is professionally presented, clear and shows an excellent understanding of the concepts and practices required.
60% <= mark< 70%	B	A working application is demonstrated which clearly shows good consideration of reuse, maintenance and usability issues and corresponds well with the design and analysis presented in the report. The report documents this comprehensive design process and is presented clearly and demonstrates a good understanding of the concepts and practices employed in development of the application.
50% <= mark< 60%	C	A working application is complemented by a comprehensive report which shows that the design was performed with some care; the design and implementation correspond closely. Some consideration of usability and reuse issues are demonstrated. The report demonstrates some understanding of the concepts required for completion of the assignment.
40% <= mark< 50%	D	Mainly working application (minor elements may be troubled), complemented by a report that shows that an analysis and design process has been followed. Implementation will match the design produced. Report covers the requirements as listed in the main assignment text and demonstrates a passable understanding of appropriate object concepts.
0% <= mark< 40%	E	Report shows erroneous design process, or an application fails to work or design work carried out but doesn't match code, or other major problem is present.

[1] The Poseidon UML tool is available as a free to use (for non commercial purposes) UML CASE tool available from Gentleware (http://www.gentleware.com)

Appendix C – part 4 Covering Letter for the Team of Evaluators

Alan Hayes
Director of Teaching
Department of Computer Science
University of Bath
BA2 7AY

26th April 2011

Dear

Thank you for agreeing to help with this research and for returning your mark sheets and comments – it is greatly appreciated. I thought that it would be timely to remind you about the main objectives of this research. The broad theme is in the area of automated assessment. I have developed a tool that analyses a student submission and provides formative feedback to the student as a consequence of this analysis. The submission consists of a design diagram (UML) and a source code implementation (java). I am now in the process of evaluating the effectiveness of the comments generated by this tool and it is this stage that I am asking for your help. I want to compare and evaluate the comments generated by my tool with those generated by a set of academic colleagues.

The evaluation takes place in two phases. Phase 1 is now completed and involved the collection of typical expert marker feedback comments. It involved you looking at a number of (anonymised) student submissions and asked you to provide the written formative feedback that you would ideally have given to the students to help them with their learning. The project is now entering Phase 2 and involves you evaluating the formative feedback comments generated by my assessment tool. More details on this second phase are provided below in addition to an indicative timescale.

I hope all is clear but if not, please get back to me.

Many thanks once again for your support.

Regards

Alan

Indicative Timescales

Activity	Completion Date
Phase 2	
Comments sent to colleagues for evaluation	**April 26th 2011**
Evaluations returned to Alan	**May 27th 2011**

Phase 2 – Evaluating Formative Feedback Comments

Please find attached the following:-

5) 10 sets of formative feedback comments.
6) 10 questionnaire forms for recording your evaluation of each set of comments.

The 10 sets of feedback comments have been generated through an analysis of student submitted coursework. Some of the sets will have been generated by one member of the marking team whilst some sets will have been generated by my marking tool. You have been randomly allocated a mixture of both human generated and tool generated comments.

The students submitted their coursework as a component of an introductory undergraduate unit/module in software development. They were asked to produce a UML diagram based upon their analysis of a given scenario. They were also required to implement their design. For many students it will have been their first experience of developing systems using object oriented methods. Consequently, they will have made the typical mistakes of novice developers. It is important that the students are not only supported in developing a strong understanding of object oriented concepts but that they also understand the software development process and in particular the link between a design and its implementation.

For each of the 10 sets of formative feedback comments please complete an evaluative questionnaire. The questionnaire consists of 14 statements and you are asked to consider how each of the 14 statements applies to each of the 10 sets of formative feedback comments. When considering a set of comments please read the set in its entirety before considering the applicability of the 14 statements.

Checklist

9) You have been sent 10 sets of formative feedback comments.
10) You have been sent 10 evaluative questionnaire forms.
11) Please complete one questionnaire per comment set.
12) Please can you return the completed mark sheet to me either by hard copy:-

Alan Hayes
Director of Teaching,
Department of Computer Science,
University of Bath,
Bath BA2 7AY.
Or electronically to

a.hayes@bath.ac.uk

Appendix C – part 5 Questionnaire Used by the Evaluative Team to Evaluate Formative Feedback Comments

Evaluation of Formative Feedback Comments

You have been sent 10 sets of formative feedback comments produced as a consequence of grading 10 separate student coursework submissions. Each submission consisted of a student design diagram (UML) and an associated implementation (java source code). The assignment brief was similar to that which you looked at during phase 1. The learning outcomes being assessed were:-

- Demonstrate a good understanding of object concepts such as encapsulation, abstraction, inheritance and polymorphism.
- Create class definitions that model real world systems.
- Create robust software which employs object concepts and techniques.
- Use an object oriented programming language to achieve a stated task.

Below is a set of 14 statements and an associated 5-point Likert scale. Please consider how each of the 14 statements applies to each of the 10 sets of formative feedback comments. When considering a set of comments please read the set in its entirety before considering the applicability of the 14 statements.

Comment Set Reference :					
Comment	Strongly Agree	Agree	Neither Agree nor Disagree	Disagree	Strongly Disagree
Criterion of Quality					
1. The comments contained in this set are clear.					
2. The comments contained in this set are concise.					
3. The set of comments provide sufficient detail in order for a student to know what concept or issue is being fed back upon.					
4. The set of comments provide sufficient detail in order for a student to know what further work they need to undertake.					
5. The set of comments will help the student with his/her learning					
Criterion of Relevance					
6. The comments contained in this set are relevant for this type of assignment brief and the associated indicative learning outcomes.					
7. The comments contained in this set address important areas of strength found in the student submission that is considered to be of significance.					
8. The comments contained in this set address important areas of weakness found in the student submission that is considered to be of significance.					
9. It is clear which concepts the comments in this set are addressing.					
10. The comments in this set will help the student improve his/her solution.					

Criterion of Coverage					
11. This set of comments, when viewed in its entirety, fully encapsulates **all** pertinent feedback needed for the student to recognise where there are areas of strength in the submission.					
12. This set of comments, when viewed in its entirety, fully encapsulates **all** pertinent feedback needed for the student to recognise where there are areas of weakness in the submission and where further learning is required.					
13. This set of comments would provide a useful enhancement to the type of comments that I gave during stage 1 of this evaluation.					
14. This set of comments would have been sufficient to replace the type of comments that I gave during stage 1 of this evaluation.					

Appendix D

Design of the Evaluative Questionnaires

Adopting a Likert scale for questionnaires poses many questions. These include:

- The number of points on the scale.
- The format of the scale.
- Whether or not a mid-point should be included on the scale.
- Interpretation of Likert data

This appendix discusses these issues in detail and the rationale for adopting a 5-point Likert scale (and by implication the inclusion of a mid-point) with named points (*strongly agree, agree, neutral, disagree, strongly disagree).*The median and mode were chosen for describing and interpreting the returns in recognition of the ordinal nature of Likert data.

1 The Number of Points on the Scale

This section discusses issues that were taken into consideration regarding the number of points to be adopted in the Likert scale. It makes a distinction between the sensitivity of the scale and the reliability of the resultant data.

The purpose of the scale is to allow a respondent to express both the direction of an opinion (for example agreeing either positively or negatively with a given statement) and an indication of the strength of agreement/disagreement with the presented statement (for example strongly agreeing or agreeing). The number of points on the scale enables the respondent to indicate the strength of agreement/disagreement. Cummins and Gullone (2000) report that the empirical literature on Likert scales supports the view that as the number of points on the scale increase, so too does the sensitivity of the scale. However, they make a distinction between the sensitivity of the scale and the reliability of the resultant data. They report upon the work of Lissittz and Green (1975) which found that the reliability of the scale increased from the adoption of a 2-point to a 5-point scale and note the work of McKelvie (1978) which found no differences in inter-rater

reliability between 5, 7 and 11 point scales. However, care must be taken when presenting respondents with more than one Likert scale. Guy and Norvell (1977) report that when respondents are presented with more than one Likert scale, a change in the number of points on the scales (in their case it was a 4-point scale without a neutral point and a 5-point scale which included one) can make a significant difference to the way a person responds.

Thus, there is a trade-off between the number of points and the reliability of the resultant data. The literature suggests that a 5 point scale is the minimum number of points required to avoid the scale itself inducing unreliable data from the respondents.

2 The Format of the Scale

This section discusses those issues that were taken into consideration regarding the naming of the points on the Likert scale. The respective benefits of naming all points on the scale as opposed to just the end-points are highlighted.

There are at least two types of format to consider when adopting a Likert scale. The first is where all points on the scale are named and defined and the second is where only the end-points are named. The adoption of the former is particularly challenging when larger scales are adopted and consequently represent a potential hindrance to the adoption of larger scales. Cummins and Gullone (2000) concluded that the "addition of category names to Likert scales not only detracts from the interval nature of the scale but also makes it difficult to generate expanded choice formats." Dixon *et al.* (1984) addressed the question of whether or not the format adopted had an influence upon the resultant data. They reported the results of applying both types of formats to 121 participants. They concluded that there was no significant difference in the data generated and that participants did not indicate a preference for either type of format. Goeb *et al.* (2007) note that,

for a 5-point Likert scale, grades are usually named with *strongly agree, agree, neutral, disagree. strongly disagree*

In summary this section discussed those issues that were taken into consideration regarding the naming of the points on the Likert scale. The respective benefits of naming all points on the scale or just the end-points were highlighted. The naming convention for 5-point Likert scales was introduced. The questionnaires adopted for this research were 5-point Likert scales with named points of *strongly agree, agree, neither agree nor disagree, disagree, strongly disagree*.

3 The Inclusion of a Mid-point in the Likert Scale

This section discusses the issue of the adoption of a mid-point in a Likert scale.

The provision of a mid-point enables the respondent to submit a neutral response to a given statement in both direction and strength. Matel and Jacoby (1972) summarise the dilemma of whether or not to include such a mid-point. Their argument against non inclusion is that it provides the respondent with "too easy and attractive an escape for respondents who are disinclined to express a definite view." Their argument for inclusion is that in forcing respondents into an agree or disagree format it is likely to cause difficulty for many respondents. Furthermore they argue that it is also likely to produce results that are "... less realistic and more misleading than is true when an intermediate reply is provided for." They note however, that as the number of points on the scale increase the use of the mid-point by the respondent decreases. They note the importance of this result in designing the construction of a Likert scale. Their advice is that if the researcher wishes to minimise the respondents usage of the mid-point then either an even-number scale should be used or an odd-numbered scale that contains many points.

In summary, this section has discussed those issues that are pertinent to whether or not to include a mid point in a Likert scale. The inclusion of a mid-point in order to produce results that are more realistic and less misleading has been highlighted. Therefore it was decided to include a mid-point in the Likert questionnaires used in this research.

4 Interpretation of Likert Data

This section discusses those issues that are pertinent to the interpretation of data that has been collated via a questionnaire that has adopted the Likert scale. The issue of ordinality of the scale and its implications for the range of statistical tests that can be conducted on resultant data is discussed.

There is no common standard accepted by the scientific community for the correct interpretation and analysis of data measured using a Likert scale (Goeb *et al.* 2007). However, both Harvey (1998) and Goeb *et al.* (2007) advocate that from a methodological perspective data collected through the adoption of a Likert scale should be considered to be ordinal, the former arguing that this is the case because it cannot be assumed that the respondent interprets that the difference on the scale between agreeing and strongly agreeing is the same as that between agreeing and being undecided.

For data returned through a Likert scale Harvey *et al.* (1998) advocate the median or mode should be adopted (and not the mean). They recommend that the mode should be used when describing the data and that the median should be used when calculating inferences. They also advocate the use of the median and the adoption of non-parametric methods to investigate differences between comparable groups. Frigon and Mathews (1997) reported that such methods are frequently used when the "standard assumptions of classical statistics are known not to be met". Diamond and Jeffries (2001) noted in particular that non-parametric

methods should be used when the data being analysed does not conform to the central limit theorem.

In summary, this section has discussed the ordinal nature of the Likert scale and its implications for undertaking an analysis of its resultant data. The literature advocates the use of the median and the adoption of non parametric techniques when analysing inferences within the data set and this advice was followed in this research.

Appendix E

Overview of the Statistical Tests Deployed

This appendix discusses the statistical techniques adopted in the evaluation of this research. Section 1 discusses the use of a Z-test and Gwet's AC1 coefficient when applied to the summative grades returned by the team of markers. Section .2 discusses the use of Gwet's AC2 statistic in the context of analysing Likert data that was returned in the evaluation questionnaires. Section .3 discusses the use of the non parametric sign and Mann-Whitney U tests to compare the questionnaire returns for the tool-generated comments with those that were human-generated.

The aim of the evaluation process was to undertake a comparison between tool and human generated comments. This process required three experiments to take place:-

- An experiment to test for significant differences between summative grades generated by a team of markers.
- An experiment to test for significant differences between members of a team of evaluators who had rated formative feedback comments.
- An experiment to test for significant differences in the evaluative ratings for the tool-generated comment when compared to those that were human-generated.

The statistical tests adopted for each of these three cases are outlined in the sections below.

1Significant differences between Summative Grades.

Two statistical tests were deployed to test for significant differences in the summative grades produced by the marking team. The tests involved the calculation of a Z score and Gwet's (2010) AC1 coefficient.

The calculation of the Z score is described in (Diamond and Jeffries 2001) as follows:

Z= (observation – mean)/standard deviation.

Its calculation requires that the population mean and standard deviation are known. In the context of this experiment these were known. Hence, it was possible to calculate a Z score for each assignment that each marker had graded. In the context of this research , the observation is the assignment grade (percentage) produced by the individual marker whilst the mean and standard deviation is calculated from all the grades (percentage) for that assignment.

Having produced the Z score a Z-test was undertaken. Critical values for a two-sided 95% confidence interval are -1.96 and 1.96.

Gwet's (2010) AC1 coefficient measures the extent to which multiple raters agree when they have analysed data and classified it into several non-overlapping categories. Haley *et al.* (2008) report upon the emergence of AC1 as an inter-rater statistic to replace the established Cohen's (1960) Kappa statistic. They report concerns over the accuracy of Kappa. In particular, Hayley *et al.* (2008) report upon an instance where

"raters agreed by as much as 97% but the Kappa statistic was close to zero, indicating no correspondence" Haley *et al.* (2008).

Additionally, Gwet (2010) reports upon the Kappa coefficient being "unstable" attributing this to an inadequate approach to compensating for the probability of chance agreements between raters. Haley (2008) presents an overview of the AC1 statistic as applied to the simpler case of two raters. This is paraphrased in the section below.

Figure AppE1 contains an example table that shows how two raters classified data into the two categories of "1" and "2". Entry A in the table represents the number of times that both raters gave a "1". Entry B is the number of times raterA gave a "2"

and raterB gave a “1”. A1 is the total number of times that raterA gave a “1” and B2 is the total number of times that raterB gave a “2”. N is the total number of observations.

Rater B	Rater A		
	1	2	Total
1	A	B	B1=A+B
2	C	D	B2=C+D
Total	A1=A+C	A2=B+D	N

Figure AppE1 Distribution of Subjects by Rater and Response Category

The probability that the two raters are in accord is known as the probability of agreement, $p_{a,}$ (Gwet 2010) and can be calculated by the formula

$$p_a = (A + D)/N$$

Cohen (1960) recognised that the probability of agreement between the two raters needs to be adjusted to take into account the possibility of the two raters agreeing on a classification merely by chance. Gwet (2010]) refers to this as the probability of the ‘expected chance agreement rate, p_e . The calculation of p_e for Kappa is given by:

$$p_e = (A1/N \times B1/N) + (A2/N \times B2/N)$$

Cohen’s (1960) Kappa coefficient, k, is subsequently defined as

$$K = (p_a - p_e) / (1 - p_e)$$

It is the consideration of p_e that Gwet (2010) argues is the cause for the instability in the Kappa statistic.

The AC1 statistic is given by the following equation

$$AC1 = (pa - p_{e1}) / (1- p_{e1})$$

where

$p_{e1} = 2p_1 (1-p_1)$ and $p_1 = ((A1+B1)/2) / N$

and

AC1 = the first order agreement coefficient

p_{e1} = the chance agreement probability

p_1 = the approximate chance that a rater classifies a subject into category 1

A1 = the number of times a rater A classifies a subject into category 1

B1 = the number of times a rater B classifies a subject into category 1

A = the number of times both raters classify a subject into category 1

D= the number of times both raters classify a subject into category 2

p_a = the overall probability of agreement

The formula for calculating the AC1 for he generalised case (more than 2 raters) is given by

$$AC1 = (pa - p_{e2}) / (1- p_{e2})$$

Where

$$Pa = \frac{1}{n} \sum_{i=1}^{N} \{ \sum_{q=1}^{Q} (r_{iq} (r_{iq} -1)/ r(r-1) \}$$

and

$$p_{e2} = \frac{1}{Q-1} \sum_{q=1}^{Q} \pi_q(1-\pi_q)$$

and

$$\Pi_q = \frac{1}{n}\sum_{i=1}^{n}\frac{r_{iq}}{r}$$

p_a = the overall probability of agreement

p_{e2} = the chance agreement probability

r_{iq} = the number of raters who classified the ith object into the qth category. The index i ranges from 1 to n and q ranges from 1 to Q.

n = the number of objects rated

Q = the number of categories in the rating scale

r = the total number of raters

Π_q = the probability that a rater classifies an object into category q.

A Worked Example of Kappa vs. AC1

Haley (2008) provides an example of how skewed data can result in an unreliable Kappa statistic. The example data used was

Rater B	Rater A		Total
	1	2	
1	45	5	50
2	5	45	50
Total	50	50	100

Rater B	Rater A		Total
	1	2	
1	90	5	95
2	5	0	5
Total	95	5	100

The table on left shows a balanced distribution of ratings whereas the table on the left shows a skewed distribution with both raters utilising the "1" category significantly more than "2". Haley calculates both the Kappa and AC1 coefficients, tabulated below.

	Balanced Distribution	Skewed Distribution
Kappa	0.8	-0.05
AC1	0.8	0.89

Intuitively, the raters in the skewed distribution are in agreement and yet the Kappa coefficient is reporting the opposite. Consequently, the AC1 statistic was chosen in preference to Kappa as being appropriate for detecting significant differences in the summative marks returned by the marking team.

2 Significant Differences between Evaluators

Formative feedback comments were evaluated by a team of evaluators utilising a Likert 5 point scale. Likert data is ordinal as it cannot be assumed that the difference between “strongly agree” and “agree” is the same as “agree” and “neither agree nor disagree”. Gwet (2010) acknowledges that the AC1 statistic is inappropriate for evaluating the extent of agreement amongst raters for ordinal data. He proposes an extension to AC1, called AC2. The extension assigns a weight to each pair of scores. When there is full agreement (i.e. all raters classify data into the same category) the weighting adopted is 1. The magnitude of the weights associated with disagreements decreases as the gap between the scores increases (Gwet 2010). Gwet (2010) provides the following formula for the weighting function:

$$W_{kl} = 1 - (x_k - x_l)^2 \; / \; \left(\frac{MAX}{1 \le k \le q; 1 \le l \le q} \; (x_k - x_l)^2 \right)$$

where

x_k and x_l = the interval scores for category k and l respectively

W_{kl} = the weighting to be applied to category K and l respectively

The formula for the AC2 statistic for two raters is provided by Gwet (2010) as

$$AC2 = (p_{a1} - p_{e2}) / (1 - p_{e2})$$

where

$$p_{a1} = \sum_{k=1}^{q} \sum_{l=1}^{q} p_{kl}$$

and

$$p_{e2} = \frac{1}{q(q-1)} \sum_{k=1}^{q} \sum_{l=1}^{q} w_{kl} (1 - q \, \Pi_k \, \Pi_l)$$

and

p_{a1} = weighted probability of agreement

p_{e2} = weighted chance agreement probability

q = number of categories

Π_k = the probability that a rater classifies an object into category k.

The generalised formula for the AC2 statistic for multiple raters is provided by Gwet (2010) as

$$AC2 = (p_{a2} - p_{e2}) / (1 - p_{e2})$$

where

$$p_{a2} = \frac{1}{nr(r-1)} \left[\sum_{k=1}^{q} \left(\sum_{i=1}^{n} w_{kk} r_{ik} (r_{ik} - 1) \right) + \sum \sum_{k \neq l} w_{kl} \left(\sum_{l=1}^{n} r_{ik} r_{il} \right) \right]$$

and

$$p_{e2} = \frac{1}{q(q-1)} \sum_{k=1}^{q} \sum_{l=1}^{q} w_{kl} (1 - q \, \Pi_k \, \Pi_l)$$

and

p_{a2} = weighted probability of agreement

p_{e2} = weighted chance agreement probability

q = number of categories

Π_k = the probability that a rater classifies an object into category

r = the number of raters

n= the number of subjects being categorized.

3 Comparing Tool-Generated with Human-Generated Comments

Diamond and Jeffries (2001) report that a non-parametric one-sample sign test can be used when comparing the median of a sample with the population median and for when the data does not follow a normal distribution. The principle behind

this test is in recognising that if the median of the population were calculated half of the observations will lie above the median and half below it. If there was no evidence that the sample was no different to the population it would be expected that about half of the observations in the sample would lie above the **population median** and half below it. If the sample is genuinely different to the population, the proportion of observations above the population median would be markedly greater or lower than 0.5. (Diamond and Jeffries 2001). The technique involves calculating the proportion of the sample whose values lies above the population median, p_m, as follows for the sample under test:

$$p_m = \frac{NumberofScoresIntheSampleAbovethePopulationMedian}{TotalNumberOfScoresInThePopulation}$$

A Z test statistic can then be calculated via the formula

$$Z = p_m - \Pi_m \frac{p_m - \Pi_m}{\sqrt{\left[\frac{\Pi_m(1-\Pi_m)}{n}\right]}}$$

Where

p_m = proportion of the sample that lies above the population median

Π_m = the proportion of the population that lies above the population median (by definition this is 0.5)

The null hypothesis is

H_0 = the sample comes from a population with half the observations above the population median.

The alternative hypothesis is

H_a = the sample does not come from a population with half the observations above the population median.

Critical values for a two-sided 95% confidence interval are -1.96 and 1.96.

The Mann-Whitney U technique is used to test for differences between two independent groups (Pallant 2007). Its use is advocated by Harvey (1998) when comparing the medians of the two groups. In the case of this research we had a set of Likert scores for human-generated comment and a set of Liker scores for those that were tool-generated. The technique involves ranking the two groups and then evaluating whether the ranks differ significantly (Pallant 2007). Ranking involves initially collating the two groups together and producing one ordered list, starting with the smallest Likert score and finishing with the highest. This list is then ranked starting with a rank value of 1 and incrementing until the list is exhausted. Where the likert scores are the same value and have the same rank an average of the rank values is taken. The method involves generating two U values, one for each group. The U value is calculated via the following formula:

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1$$

$$U_2 = n_1 n_2 + \frac{n_2(n_2+1)}{2} - R_2$$

Where

n_1 = the number of scores in group 1

n_2 = the number of scores in group 2

R_1 = the sum of the Ranks for group 1

R_2 = the sum of the Ranks for group 2

The Null hypothesis is given by

H_0 = There is no difference in the distribution of scores for both groups

H_a = There is a difference in the distribution of scores for both groups

$U_{critical}$,the test value for the 95% confidence interval, varies according to the size of the respective groups. The null hypothesis is rejected if the smallest value of U_1 or U_2 is less than $U_{critical}$.

4 Statistical Packages Deployed

This appendix has discussed the statistical techniques deployed during the evaluation of the tool-generated comments. The calculation of the statistical coefficients used in this evaluation was undertaken via two separate software packages. Gwet's AC1 and AC2 coefficients were calculated using the agreestat tool. This is a tool made available in June 2011 by Gwet and accessible from http://www.agreestat.com. The Mann-Witney U coefficients were calculated by IBM's SPSS tool (version 19). The Z-score was produced manually via an Excel spreadsheet.

Appendix F

ANOVA test results for the percentage grades received for three, randomly chosen, student submissions.

This appendix presents an ANOVA analysis of the percentage grades received for three, randomly chosen student submissions. The analysis was undertaken using the data analysis package contained within Microsoft Excel 2010. Three tables, one per submission, present the results of undertaking an Anova:single factor analysis (alpha=0.1). The null hypothesis is:

H_0 = The summative grade from an individual marker is from the same population as that received from all markers.

The analysis produces a test statistic Fstat which is compared with Fcritical. The null hypothesis is rejected when F is greater than Fcritical. As can be seen from below this happens for markers 8 and 3 (assignments 17 and 79 respectively). Hence, the conclusion is that markers 8 and 3 have viewed the student submission differently from the rest of the markers and consequently their formative comments were removed from the remainder of the research. This is consistent with the Z test results presented in the main body of the thesis (chapter 6).

Results for Assignment 17

Ref	Ass 17	F-stat	p-value	F-critical	Include Comments Based on Ass 17
Marker 2	79	0.39	0.55	3.46	Y
Marker 3	75	1.83	0.21	3.46	Y
Marker 4	82	0.01	0.94	3.46	Y
Marker 5	82	0.01	0.94	3.46	Y
Marker 6	80	0.20	0.67	3.46	Y
Marker 7	84	0.08	0.78	3.46	Y
Marker 8	**94**	**4.41**	**0.07**	**3.46**	**N**
Marker 9	85	0.22	0.65	3.46	Y
Marker 10	81	0.07	0.80	3.46	Y

Results for Assignment 79

Ref	Ass79	F-stat	p-value	F-critical	Include Comments Based on Ass 79
Marker 2	82	0.00	0.95	3.59	Y
Marker 3	**60**	**3.79**	**0.09**	**3.59**	**N**
Marker 4	86	0.19	0.68	3.59	Y
Marker 5	no return				n.a.
Marker 6	73	0.57	0.47	3.59	Y
Marker 7	92	0.97	0.36	3.59	Y
Marker 8	89	0.50	0.50	3.59	Y
Marker 9	82	0.00	0.95	3.59	Y
Marker 10	86	0.19	0.68	3.59	Y

Results for Assignment 182

Ref	Ass 182	F-stat	p-value	F-critical	Include Comments Based on Ass 182
Marker 2	85	0.00	0.97	3.46	Y
Marker 3	75	0.91	0.37	3.46	Y
Marker 4	91	0.42	0.54	3.46	Y
Marker 5	68	2.74	0.14	3.46	Y
Marker 6	81	0.13	0.73	3.46	Y
Marker 7	93	0.71	0.42	3.46	Y
Marker 8	94	0.89	0.37	3.46	Y
Marker 9	79	0.31	0.59	3.46	Y
Marker 10	95	1.09	0.33	3.46	Y

Appendix G

Mann Witney U test results for Likert ratings received for human- and tool-generated formative feedback comments.

The tables below present the results of using a Mann Witney U test to compare Likert grades for human-generated formative assessment comments with those that were tool generated. The tool used to undertake this analysis was StatsDirect (http://www.statsdirect.com accessed 02/01/2013). The tables present the output of one sided (upper and lower) and two sided tests.

The null hypothesis is:

H_0: The distribution of Likert scores is the same across the human-generated comments as it is for the tool-based comments.

The likert scaling used was:

Likert Scoring		
5		Strongly Agree
4		Agree
3		Neither Agree nor Disagree
2		Disagree
1		Strongly Disagree

The results show that the null hypothesis is rejected and that tool-generated comments are ranked higher than human-generated.

Mann-Whitney U test

Observations (x) in Q1 human = 31 median = 4 rank sum = 720
Observations (y) in Q1 tool = 32 median = 4.5
U = 224 U' = 768

Normalised statistic = -4.092119 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P < 0.0001 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -1
CI = -1 to 0

Mann-Whitney U test

Observations (x) in Q2 human = 31 median = 4 rank sum = 1162.5
Observations (y) in Q2 tool = 32 median = 4
U = 666.5 U' = 325.5

Normalised statistic = 2.471769 (adjusted for ties)
Lower side P = 0.9933 (H_1: x tends to be less than y)
Upper side P = 0.0067 (H_1: x tends to be greater than y)
Two sided P = 0.0134 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = 1
CI = 0 to 1

Mann-Whitney U test

Observations (x) in Q3 human = 31 median = 3 rank sum = 679
Observations (y) in Q3 tool = 32 median = 4
U = 183 U' = 809

Normalised statistic = -4.625287 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P < 0.0001 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -1
CI = -2 to -1

Mann-Whitney U test

Observations (x) in Q4 human = 31 median = 2 rank sum = 660
Observations (y) in Q4 tool = 32 median = 4
U = 164 U' = 828

Normalised statistic = -4.796196 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P < 0.0001 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -2

CI = -2 to -1

Mann-Whitney U test

Observations (x) in Q5 human = 31 median = 3 rank sum = 710.5
Observations (y) in Q5 tool = 32 median = 4
U = 214.5 U' = 777.5

Normalised statistic = -4.125512 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P < 0.0001 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -1
CI = -2 to -1

Mann-Whitney U test

Observations (x) in Q6 human = 31 median = 4 rank sum = 739.5
Observations (y) in Q6 tool = 32 median = 4
U = 243.5 U' = 748.5

Normalised statistic = -3.766679 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P = 0.0002 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -1
CI = -1 to 0

Mann-Whitney U test

Observations (x) in Q7 human = 31 median = 2 rank sum = 597.5
Observations (y) in Q7 tool = 32 median = 4
U = 101.5 U' = 890.5

Normalised statistic = -5.654318 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P < 0.0001 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -2
CI = -3 to -1

Mann-Whitney U test

Observations (x) in Q8 human = 31 median = 4 rank sum = 679
Observations (y) in Q8 tool = 32 median = 5
U = 183 U' = 809

Normalised statistic = -4.594558 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P < 0.0001 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -1
CI = -2 to -1

Mann-Whitney U test

Observations (x) in Q9 human = 31 median = 3 rank sum = 647
Observations (y) in Q9 tool = 32 median = 4
U = 151 U' = 841

Normalised statistic = -5.140455 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P < 0.0001 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -1
CI = -2 to -1

Mann-Whitney U test

Observations (x) in Q10 human = 31 median = 3 rank sum = 695
Observations (y) in Q10 tool = 32 median = 4
U = 199 U' = 793

Normalised statistic = -4.369207 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P < 0.0001 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -1
CI = -2 to -1

Mann-Whitney U test

Observations (x) in Q11 human = 31 median = 2 rank sum = 550
Observations (y) in Q11 tool = 32 median = 4
U = 54 U' = 938

Normalised statistic = -6.265161 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P < 0.0001 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -2
CI = -3 to -2

Mann-Whitney U test

Observations (x) in Q12 human = 31 median = 2 rank sum = 596
Observations (y) in Q12 tool = 32 median = 4
U = 100 U' = 892

Normalised statistic = -5.632564 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P < 0.0001 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -2
CI = -3 to -2

Mann-Whitney U test

Observations (x) in Q13 human = 31 median = 3 rank sum = 596
Observations (y) in Q13 tool = 32 median = 4
U = 100 U' = 892

Normalised statistic = -5.637128 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P < 0.0001 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -2
CI = -2 to -1

Mann-Whitney U test

Observations (x) in Q14 human = 31 median = 2 rank sum = 582.5
Observations (y) in Q14 tool = 32 median = 4
U = 86.5 U' = 905.5

Normalised statistic = -5.772615 (adjusted for ties)
Lower side P < 0.0001 (H_1: x tends to be less than y)
Upper side P > 0.9999 (H_1: x tends to be greater than y)
Two sided P < 0.0001 (H_1: x tends to be distributed differently to y)

95% confidence interval for difference between medians or means:
K = 354 median difference = -2
CI = -3 to -1